UNIVERSITY OF CALIFORNIA

Santa Barbara

# Accelerated Algorithms for Stochastic Simulation of Chemically Reacting Systems

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in Computer Science

by

Jin Fu

Committee in Charge:

Professor Linda Petzold, Chair

Professor John Gilbert

Professor Frank Doyle

December 2014

The dissertation of Jin Fu is approved.

---

Professor John Gilbert

---

Professor Frank Doyle

---

Professor Linda Petzold, Committee Chairperson

December 2014

Accelerated Algorithms for Stochastic Simulation of Chemically Reacting Systems



by

Jin Fu

# Acknowledgements

I would like to thank my parents, who give me continuous support during the years when I was in UCSB.

I would like to thank my academic adviser, Professor Linda Petzold, who provides me the opportunity to study in her lab and makes the contribution in this thesis possible.

I would like to thank the other committee members, Professor John Gilbert and Professor Frank Doyle, who give me valuable suggestions in my major area exam, thesis proposal and defense.

I will thank my labmates as well. You give me many help for my work. And more importantly, you give me a good time in UCSB.

# 致谢

首先，我要感谢我的父母，你们的关心和支持是我攻读博士学位最大的动力。和你们的视频聊天总是那么开心。让我体验到了家的感觉。看到你们生活的无忧无虑，我真的非常高兴。

我要感谢我的导师,Linda Petzold 教授。您给了我来实验室攻读博士学位的机会，也在我的学习和研究工作中给了我非常多的帮助和指导。我非常庆幸能遇到像您这么好的导师。

我还要感谢我答辩组中的另两位教授，Jhon Gilbert 教授和 Frank Doyle 教授。你们给我提出的宝贵建议对我的研究工作帮助非常大。

最后，我要感谢实验室的所有同学。和你们在一起的日子是轻松快乐的。时光匆匆，如今我们要分开了，希望你们也都能顺利实现自己的目标。

# Curriculum Vitæ

Jin Fu

## EDUCATION

Bachelor of Engineering in Mechanics, Peking University, July 2004

Bachelor of Science in Math, Peking University, July 2004

Master of Engineering in Mechanics, Beihang University, March 2008

Master of Science in Computer Science, University of California, Santa Barbara, June 2013

Doctor of Philosophy in Computer Science, University of California, Santa Barbara, December 2014 (expected)

## PROFESSIONAL EMPLOYMENT

2006-08: Teaching Assistant, School of Science, Beihang University

2008-14: Research Assistant, Department of Computer Science, University of California, Santa Barbara

## PUBLICATIONS

"The Time Dependent Propensity Function for Acceleration of Spatial Stochastic Simulation of Reaction-Diffusion Systems," *Journal of Computational Physics*, 274 (2014), 524–549.

"Time Dependent Solution for Acceleration of Tau-Leaping," *Journal of Computational Physics*, 235 (2013), 446-457.

"Automatic Identification of Model Reductions for Discrete Stochastic Simulation," *Journal of Chemical. Physics*, 137 (2012), 034106.

"StochKit2: Software for Discrete Stochastic Simulation of Biochemical Systems with Events," *Bioinformatics*, 27 (2011), 2457-2458.

"Michaelis-Menten Speeds up Tau-leaping Under a Wide Range of Conditions," *Journal of Chemical Physics*, 134 (2011), 134112.

## FIELDS OF STUDY

Major Field: Computational Science and Engineering

# Abstract

## Accelerated Algorithms for Stochastic Simulation of Chemically Reacting Systems

Jin Fu

Stochastic models are widely used in the simulation of biochemical systems at a cellular level. For well mixed models, the system state can be represented by the population of each species. The probabilities for the system to be in each state are governed by the Chemical Master Equation (CME), which is generally a huge ordinary differential equation (ODE) system. The cost of solving the CME directly is generally prohibitive, due to its huge size.

The Stochastic Simulation Algorithm (SSA) provides a kinetic Monte Carlo approach to obtain the solution to the CME. It does this by simulating every reaction event in the system. A great many stochastic realizations must be performed, to obtain accurate probabilities for the states. The SSA can generate a highly accurate result, however the computation of many SSA realizations may be expensive if there are many reaction events. Tau-leaping is an approximate algorithm that can speed up the simulation for many systems. It advances the system with a selected stepsize. In each step, it directly samples the number of reaction events in each reaction channel, which yields a faster simulation than SSA. The error in tau-leaping is controlled by selecting the stepsize properly.

We have developed a new, accelerated tau-leaping algorithm for discrete stochastic simulation that make use of the fact that exact (time-dependent) solutions are known for some of the most common reaction motifs (subgraphs of the network of chemical species and reactants). This idea can be extended to spatial stochastic simulation, by treating the diffusion network as a special motif for which there is an exact time dependent solution. We describe the well-mixed and spatial stochastic time dependent solution algorithms, along with numerical experiments illustrating their effectiveness.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Ordinary differential equation (ODE) models are widely used in the simulation of chemical systems where all chemical species are present with large population. For the simulation of biochemical systems inside a living cell, however, the population of some chemical species may be so small that stochastic fluctuations become important [1, 2, 3]. For these systems, a discrete stochastic model is more appropriate. In a discrete stochastic model, the system state is no longer deterministic at a time $t > t_0$, where $t_0$ is the initial time. Instead, it could be in one of several possible states, with certain probabilities. The dynamics of the probabilities is governed by the Chemical Master Equation (CME) [4].

The CME is a huge ODE when a system has many possible states. Thus it is usually extremely expensive to solve. However, Monte Carlo simulation provides a different way to find the solution of the CME. The stochastic simulation algorithm (SSA) [5, 4] is commonly used to simulate a stochastic reaction system. The SSA samples when the next reaction event occurs and which reaction will fire. Then it advances the system to that time and updates the system state by firing the sampled reaction event.

The SSA is exact, in the sense that each simulation is a realization of the CME. As the number of stochastic realizations goes to infinity, their statistics approach the probability density vectors (PDVs) that are the solutions to the CME. Alternative formulations of the SSA include the optimized direct method (ODM) [6], the composition-rejection method [7], the rejection-based SSA (RSSA) [8] and the next reaction method (NRM) [9].

Typically, a great many (hundreds of thousands to millions) of simulations are required to obtain a good approximation to the PDVs. At the same time, each realization can be quite expensive for exact algorithms. This is because every reaction event in the system must be sampled. Approximate algorithms have been developed to overcome this limitation. Tau-leaping [10] is an approximate algorithm that can for many systems take time steps that are considerably larger than the time to the next reaction (i.e. the SSA timestep). It accomplishes this by allowing multiple reaction events to fire during a timestep as long as these reactions do not change the system dramatically, i.e. the change of each species during a step is small compared with its population. The stepsize

for tau-leaping can become constrained, however, for systems with fast reactions that involve at least one species that is present in very small population [11].

One way to accelerate both SSA and tau-leaping for such stiff systems is to make use of a stochastic quasi-steady-state assumption. The quasi-steady-state assumption is a widely used strategy to handle systems that have different time scales, for both ODE [12] and SSA [13, 14, 15]. The essence of this strategy is to divide the system into fast and slow subsystems. If the fast subsystem can reach a stochastic quasi-steady-state in a very short time, then we can use the quasi-steady-state as an approximation of the fast variables during a step of the slow subsystem. One can also apply the quasi-steady-state assumption in tau-leaping [11]. However, we must be careful when using this assumption. If the fast subsystem cannot reach a steady distribution rapidly enough, the quasi-steady-state assumption may introduce too much error into the simulation.

To avoid these errors, we can use the *time dependent solution* [16] rather than the quasi-steady-state. The idea of using the time dependent solution to speed up a discrete stochastic simulation has been applied via a splitting method in [17]. That method first partitions the reactions into subgroups such that some of them have analytical solutions, which can be used to directly sample the state of the subsystem at any given time if reactions outside the subsystem are kept silent. Then the method advances the system by advancing each subsystem separately in a given order with some stepsize. Since it can directly sample the state without sampling individual reaction events for those subsystems that have analytical solutions, it is more efficient than SSA if these

subsystems contain many reaction events. However, it does not handle non-catalytic bimolecular reactions with the time dependent solution, or provide a stepsize selection strategy. The adaptive tau-leaping method addresses these two issues. It approximates the number of firings for bimolecular reactions in each step [10] and it also has an adaptive stepsize selection algorithm [18].

In Chapter 2 of this thesis, we introduce a methodology to apply the time dependent solution in a tau-leaping framework. Thus the analytical solution can be used to approximatie the solution of bimolecular reactions such as $S_1 + S_2 \rightarrow something$ within a tolerance. The new algorithm inherits the adaptive stepsize selection strategy of [18] naturally as well. This algorithm has been implemented in the software package STOCHKIT 2 [19].

Generally speaking, the time dependent solution is not easy to derive for an arbitrary network motif. However, for some common motifs we do have time dependent solutions. These solutions can be used to improve the performance of tau-leaping for some widely used models like the enzyme-substrate model.

The previous methods work for well-mixed models. In a spatially inhomogeneous setting, the volume is divided into subvolumes. In each subvolume, the well-mixed assumption is applied to reactions. Diffusive transfers between adjacent subvolumes are modeled as monomolecular reactions. The master equation for the inhomogeneous system is called the reaction diffusion master equation (RDME) [20]. SSA algorithms can be applied in the inhomogeneous setting as well (ISSA). The most popular formulation

of the ISSA is the next subvolume method (NSM) [21]. The NSM uses a similar idea as the NRM. It generates the next event's time for every subvolume. Here an event could be a reaction event or a diffusion event. In a simulation step, the NSM picks the subvolume with the smallest time to the next event, and samples an event in it. Since the NSM can find the subvolume where the next event occurs in $O(\log N)$ time, where $N$ is the number of subvolumes, it has better performance than the direct method when the system has many subvolumes. The NSM has been implemented in software packages such as MesoRD [22] and URDME [23].

Approximation-based methods have been developed for further speeding up the simulation. The multinomial simulation algorithm (MSA) [24] splits the reaction and diffusion processes. In each step it samples the next reaction time based on the current state, then it samples the position of every particle using multinomial distributions, which no longer need to track every diffusion event as the exact methods do. After the diffusion process sampling, the MSA updates the system by firing a sampled reaction. The diffusive finite state projection algorithm (DFSP) [25] employs a similar idea but it allows multiple reaction events to fire in one step. It uses SSA to simulate the reaction process in each subvolume independently in each step. The diffusion process is sampled by solving the diffusion master equation with truncated states. Hybrid methods are another approach for simplifying the simulation. In a hybrid method, the reactions (both chemical reactions and diffusive jumps) are partitioned into several parts. Different parts are treated with different methods. The software package URDME [23] includes

an adaptive hybrid method [26] along with NSM and DFSP, for stochastic reaction-diffusion processes.

In MSA, DFSP and the adaptive hybrid method, the reaction and diffusion processes are decoupled in every step. These methods sample the next reaction time based on the current state, i.e. by assuming that the system state does not change between adjacent chemical reaction events. However, this is an approximation because molecules will be diffusing during that time. In Chapter 3 of this thesis we present a method that uses the *time dependent propensity function* [27] to sample the reaction events. We will refer to the method as the time dependent propensity for diffusion method (TDPD).

The idea of of using the time dependent propensity in a simulation has previously been introduced, in a non-spatial form, in [9], where the NRM was extended for time varying Markov processes and some examples are provided. A non-Markov process example was discussed in that paper, where the time dependent propensity, which is a gamma distribution, yields an efficient algorithm for the simulation. In [28], the idea of using the time dependent propensity was incorporated into a hybrid method, where the time dependent propensity of discrete reactions was computed by the values generated from the continuous reactions.

The basic idea of the TDPD method is that it uses the time between adjacent reaction events as the simulation stepsize, which is the same as SSA. However, the time dependent propensity function, which is used for sampling the next reaction time

in TDPD, takes into account the change of the propensity values during a stepsize due to the diffusion process. Thus the method yields a speedup by avoiding the effort of tracking individual diffusion events, while still enjoying excellent accuracy. This algorithm has been implemented in the software package STOCHKIT 2 [19] with regular mesh in rectangular domain and in PyURDME [29] with unstructured mesh in arbitrary domain.

The remainder of the thesis is organized as follows. In Chapter 2 we introduce the algorithm that uses the time dependent solution in tau-leaping. In Chapter 3, the TDPD method is developed. Chapter 4 describes the extension of the TDPD method to an unstructured mesh.

# Chapter 2

# Time Dependent Solution for Acceleration of Tau-Leaping

## 2.1 Introduction

Tau-leaping [10] is an approximate algorithm that is faster than SSA [5, 4] for many systems. However, its stepsize can become constrained if a system has fast reactions that involve at least one species with very small population. If the population of such a species reaches a steady distribution rapidly, the stochastic quasi-steady-state assumption [13, 14, 15, 11] can be used to handle this situation. If this is not the case, we can use the *time dependent solution* [16] instead for many common motifs. In this chapter we introduce our algorithm that uses the time dependent solution to accelerate tau-leaping.

This chapter is organized as follows. In Section 2.2, we provide a brief introduction to tau-leaping with adaptive timestep selection. In Section 2.3 we derive the time dependent solution for some common network motifs. We begin with a simple example to demonstrate the tau-leaping algorithm using the time dependent solution. Then we extend the algorithm to more general cases. Numerical experiments are provided in Section 2.4, including application of the method to a realistic model of blood coagulation, and the algorithm is briefly summarized in Section 2.5. Detailed mathematical derivations are provided in the Appendix of this thesis. This work was published in *Time dependent solution for acceleration of tau-leaping* (Jin Fu, Sheng Wu, and Linda R. Petzold. J. Comput. Phys., 235:446–457, 2013).

## 2.2 Tau-Leaping

Consider a system of $N$ species $\{S_1, \ldots, S_N\}$ and $M$ reactions $\{R_1, \ldots, R_M\}$. The state vector of the system is $\boldsymbol{X} = \{x_1, \ldots, x_N\}$ which is the population of each of the species. The probability that reaction $R_i$ fires in an infinitesimal interval $dt$ is given by $a_i(\boldsymbol{X})dt$, where $a_i(\boldsymbol{X})$ is the propensity function of $R_i$. Tau-leaping advances the system in small steps; it assumes that the state vector $\boldsymbol{X}$ changes so little in each step that the propensity functions $\{a_1, \ldots, a_M\}$ can be treated as constants. Thus the number of firings in each reaction channel $R_i$ is a Poisson random number with parameter $a_i(\boldsymbol{X})\tau$, where $\tau$ is the stepsize. To advance the system, we need only to sample these Poisson random numbers and update the state vector $\boldsymbol{X}$.

Yang et al. [18] suggest a strategy to determine the stepsize. The idea is that it should be chosen so that the mean and standard deviation of the change of each species is small compared to its population. Denoting the population change of species $S_i$ as $\Delta x_i$, the stepsize as $\tau$, and the number of firings of each reaction during a step as $r_1(\tau), \ldots, r_M(\tau)$, tau leaping computes

$$\Delta x_i = \sum_{j=1}^{M} \nu_{ij} r_j(\tau),$$

where $\nu_{ij}$ is the stoichiometry of species $S_i$ in reaction $R_j$. Assuming that the reaction firings are independent during a step, the mean and variance of $\Delta x_i$ are given by

$$\mathbb{E}\Delta x_i = \sum_{j=1}^{M} \nu_{ij} \mathbb{E}(r_j(\tau)), \quad \mathrm{Var}(\Delta x_i) = \sum_{j=1}^{M} \nu_{ij}^2 \mathrm{Var}(r_j(\tau)).$$

Keeping $\mathbb{E}\Delta x_i$ and $\sqrt{\mathrm{Var}\Delta x_i}$ small (relative to the tolerance $\epsilon$) compared with $x_i$ requires [18]

$$\mathbb{E}\Delta x_i \leq \max\left(\frac{\epsilon}{g_i} x_i, 1\right), \quad \sqrt{\mathrm{Var}(\Delta x_i)} \leq \max\left(\frac{\epsilon}{g_i} x_i, 1\right), \tag{2.1}$$

where $g_i$ is a constant that depends on the highest order of the reactions which involve $S_i$ as a reactant. Solving the above inequalities yields the upper bound on $\tau$, which we will denote by $\tau_i$, for which species $S_i$ can be expected to change by less than the prescribed tolerance. The adaptive tau-leaping algorithm chooses the smallest $\tau_i$ as its stepsize.

$$\tau = \min_{1 \leq i \leq N} \tau_i \tag{2.2}$$

Over a step of size $\tau$, tau-leaping approximates the population of every species as a constant. Thus $r_i(\tau)$ is a Poisson random variable

$$r_i(\tau) \sim \mathcal{P}(a_i \tau).$$

Solving (2.1) for $\tau_i$ gives

$$\tau_i \leq \frac{\max\left(\frac{\epsilon}{g_i} x_i, 1\right)}{\sum_{j=1}^{M} \nu_{ij} a_j}, \quad \tau_i \leq \frac{\max\left(\frac{\epsilon^2}{g_i^2} x_i^2, 1\right)}{\sum_{j=1}^{M} \nu_{ij}^2 a_j} \Rightarrow \tau_i = \min\left(\frac{\max\left(\frac{\epsilon}{g_i} x_i, 1\right)}{\sum_{j=1}^{M} \nu_{ij} a_j}, \frac{\max\left(\frac{\epsilon^2}{g_i^2} x_i^2, 1\right)}{\sum_{j=1}^{M} \nu_{ij}^2 a_j}\right), \tag{2.3}$$

and substituting this into (2.2) yields the tau-leaping stepsize.

It is easy to see that tau-leaping can be substantially more efficient than SSA. However, this is only the case when it can use a stepsize over which many reaction firings would have taken place. However, if some species $S_i$ is changing rapidly, then the change in that species may be constraining the stepsize. On each timestep, the species that is constraining the stepsize is the one for which $\tau_i$ is smallest. Thus we propose to use the time dependent solution described in the next section to solve for that species in place of standard tau-leaping (provided that it occurs in one of the common network motifs for which we have a time dependent solution).

Using the time dependent solution is a natural way to remove the stepsize constraint from the limiting species. This idea can also be extended to cases where several species require a very small stepsize. Though a general solution for arbitrary motifs may not be easy to find, we do have the solution for some common motifs. The results will be shown in the next section.

## 2.3 Tau-leaping using the time dependent solution

The time dependent solution makes use of the exact analytical solution of common reaction motifs to increase the speed of tau-leaping. The splitting method [17] also uses the analytical solution of monomolecular, catalytic bimolecular, and autocatalytic reactions. It separates these reactions from the system to form subsystems that can be simulated using their analytical solutions. The time dependent solution improves on the splitting method in the following two ways.

- Applicability to non-catalytic bimolecular reactions.

  In order to use the analytical solution for a bimolecular reaction, the splitting method requires that one of its reactants has zero stoichiometry (i.e. catalytic bimolecular reaction). The time dependent solution removes this requirement by observing that if one of the reactants of a non-catalytic bimolecular reaction has a slow relative rate of change, we should be able to allow it to use the analytical solution to within some tolerance.

  This change brings new requirements to the system partitioning strategy. In the splitting method the subsystems are determined by the stoichiometry. Thus it can partition the system at the very beginning and use that partitioning throughout the simulation. However, if we allow the subsystems to include non-catalytic bimolecular reactions, the stoichiometry matrix will not be sufficient to determine the partitioning of the system. We also need the information of the dynamically

changing reaction rates. Thus the time dependent solution includes a scheme for dynamic partitioning.

- Adaptive stepsize selection

  An operator bounding analysis for the splitting method was given in [17]. For simulation purposes, it would be ideal if the analysis can generate an algorithm to adaptively select the stepsize. Here, since our partition will be more complex and our implementation of the time dependent solution is in the tau-leaping framework, making use of the adaptive stepsize selection strategy from tau-leaping [18] is a more natural and easy option for our method.

In this section we will demonstrate the use of the time dependent solution using the tau-leaping method. We begin with a simple example.

### 2.3.1 Using the time dependent solution of one species

Let us take a look at one species in particular, say $S_1$. There are reactions which either generate or consume $S_1$, as shown in Figure 2.1. We will refer to the motif illustrated in Figure 2.1 as Motif I in the following sections.

If for any reaction in the system, its reactants involve at most one $S_1$ molecule and its products also involve at most one $S_1$ molecule, then we can find the analytical solution for the population of $S_1$, under the assumption that the populations of other species can be considered as constants. This assumption is reasonable as long as we use a stepsize that can be accepted by those other species.

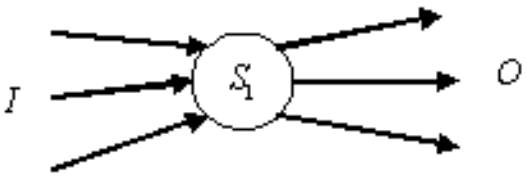



Figure 2.1: Motif I, $I$ denotes the set of reactions that generate $S_1$, and $O$ denotes the set of reactions that consume $S_1$.

Let $I$ be the set of reactions that generate $S_1$, and $O$ be the set of reactions that consume $S_1$. Denote the total propensity that an $S_1$ will be generated as

$$a_I \stackrel{\triangle}{=} \sum_{R_i \in I} a_i,$$

and the total rate that $S_1$ will be consumed as

$$c_O \stackrel{\triangle}{=} \sum_{R_i \in O} \tilde{c}_i,$$

where $\tilde{c}_i = a_i/x_1$.

The time dependent population of $S_1$ can be written as (see Appendix A.1)

$$x_1(t) \sim \mathcal{B}\left(x_1(0),\ e^{-c_O t}\right) + \mathcal{P}\left(\frac{a_I}{c_O}\left(1 - e^{-c_O t}\right)\right) \tag{2.4}$$

$$\sim \mathcal{B}\left(x_1(0),\ e^{-c_O t}\right) + \mathcal{B}\left(r_I,\ \frac{1}{c_O t}\left(1 - e^{-c_O t}\right)\right), \tag{2.5}$$

where $x_1(0)$ is the initial value of $x_1$ at the beginning of the step, and $r_I$ is the input to $S_1$, i.e. the total number of firings for reactions in $I$. $\mathcal{B}(n,\ p)$ is a binomial random number with parameters $n,\ p$. $\mathcal{P}(\lambda)$ is a Poisson random number with parameter $\lambda$. The two random variables in (2.4) and (2.5) are independent.

The corresponding output from $S_1$, i.e. the total number of firings in $O$, is given by

$$r_O(t) \stackrel{\triangle}{=} \sum_{R_i \in O} r_i(t) = x_1(0) + r_I - x_1(t)$$
$$\sim \mathcal{B}\left(x_1(0),\ 1 - e^{-c_O t}\right) + \mathcal{B}\left(r_I,\ 1 - \frac{1}{c_O t}\left(1 - e^{-c_O t}\right)\right). \tag{2.6}$$

To simulate the number of firings in each reaction channel $R_i \in O$, we distribute $r_O$ using the multinomial distribution according to the rate $\tilde{c}_i$ of each reaction $R_i$

$$\{r_i :\ R_i \in O\} \sim \mathcal{M}\left(r_O,\ \frac{\tilde{c}_i}{c_O} :\ R_i \in O\right) \tag{2.7}$$

or equivalently (see Appendix A.3),

$$r_i(t) \sim \mathcal{B}\left(x_1(0),\ \frac{\tilde{c}_i}{c_O}\left(1 - e^{-c_O t}\right)\right) + \mathcal{P}\left(\frac{\tilde{c}_i}{c_O}\left(a_I t - \frac{a_I}{c_O}\left(1 - e^{-c_O t}\right)\right)\right). \tag{2.8}$$

Here $\mathcal{M}\left(n,\ p_1, \ldots, p_n\right)$ is a multinomial random variable with parameters $n$ and $p_1, \ldots, p_n$. Now we apply this time dependent solution to accelerate tau-leaping for the simple example.

$$S_0 \underset{c_2}{\overset{c_1}{\rightleftharpoons}} S_1 \xrightarrow{c_3} S_2.$$

When the population of $S_0$ is much greater than the population of $S_1$, $S_1$ will be the species that limits the tau-leaping stepsize. Using the time dependent solution of $S_1$ we arrive at the following algorithm.

1. Use (2.3) to compute the acceptable stepsizes $\tau_i$ for every species (in this case $S_0$ and $S_1$. There is no need to compute $S_2$ because it is a pure product and it never changes any propensity function).

2. Find the smallest $\tau_i$ (Here we assume $\tau_1 < \tau_0$ for demonstration purposes, so $I = \{R_1\},\ O = \{R_2,\ R_3\}$).

3. Recompute the stepsize. In this example we need to recompute $\tau_0$ for $S_0$. We do this because the original $\tau_0$ was based on the assumption that $x_1$ is a constant during the step. Since this is no longer the case, we need to reevaluate $\tau_0$. To do this, we still try to bound the mean and variance of $\Delta x_0$ using (1). The only change is that the number of firings of $R_2$ is no longer a Poisson random variable. Instead, we have formula (2.8) for $r_2$, so both $\mathbb{E}(r_2)$ and $\mathrm{Var}(r_2)$ can be obtained explicitly and used to compute the new value for $\tau_0$. (Here we need to solve a nonlinear algebraic equation since $\mathbb{E}(r_2)$ and $\mathrm{Var}(r_2)$ contain $e^{-c_O t}$ terms. Newton iteration is a good option because the explicit formulas of the equations are known).

4. Sample the number of firings in all reaction channels except those belonging to $O$ (Sample $r_1(\tau)$ in the example). These reactions do not depend on the species for which we use the time dependent solution ($S_1$ in the example), so the original strategy in tau-leaping still works. Reactions in $I$ are sampled in this step so that we know the value of $r_I$.

5. Sample $r_O$ using (2.6) and distribute it into each channel in $O$ using (2.7). (Now $r_2$ and $r_3$ have been sampled).

6. Update the system and start the next step, or terminate if the end time of the simulation has been reached.

In some reacting systems, there can be reactions that use $S_1$ as a catalyst. For example, suppose that we add the following reaction $R_4$ to the above system

$$R_4 : \ S_1 \xrightarrow{c_4} S_1 + S_3.$$

This reaction cannot be sampled using a Poisson random number $\mathcal{P}\left(c_4 x_1\left(0\right)\tau\right)$ in the previous framework, since $S_1$ may undergo a big change during the step. This reaction does not belong to $O$, since it does not consume $S_1$. It needs to be treated as a different case.

The value of $r_4$ during a step is given by

$$r_4 \sim \mathcal{P}\left(\int_0^\tau c_4 x_1\left(t\right) dt\right).$$

Since we cannot compute the integral exactly, we will need to make an approximation. A natural choice is to use the mean value $\mathbb{E}\left(x_1(t)\right)$ instead of the exact random number $x_i\left(t\right)$, which yields

$$r_4 \approx \mathcal{P}\left(c_4 \int_0^\tau \mathbb{E}\left(x_1\left(t\right)\right) dt\right). \tag{2.9}$$

This value is capable of being sampled, since we can derive the formula for $\mathbb{E}\left(x_1\right)$ from (2.4). Thus we have a formula for the integral expression. This approximation can capture the mean value of $r_4$ accurately but its variance is smaller than the exact value of $\mathrm{Var}\left(r_4\right)$ (see Appendix A.2). This is because $\mathbb{E}\left(x_1(t)\right)$ averages $x_1(t)$, thus it loses the specific information of the trajectory. To recover the variance, we need to include

this information in the approximation. Since in Step 5 of the algorithm $x_1(\tau)$ is sampled (more precisely, we sample $r_O$, however we can get $x_1(\tau)$ by $x_1(\tau) = x_1(0)+r_I-r_O(\tau)$), it would be advantageous if we could include this information in the approximation. This yields another approximation formula:

$$\begin{aligned} r_4 &\approx \mathcal{P}\left(c_4 \int_0^\tau \left(\mathbb{E}\left(x_1\left(t\right)\right) + \frac{t}{\tau}\left(x_1\left(\tau\right) - \mathbb{E}\left(x_1\left(\tau\right)\right)\right)\right) dt\right) \\ &\sim \mathcal{P}\left(c_4 \left(\int_0^\tau \mathbb{E}\left(x_1\left(t\right)\right) dt + \frac{\tau}{2}\left(x_1\left(\tau\right) - \mathbb{E}\left(x_1\left(\tau\right)\right)\right)\right)\right). \end{aligned} \tag{2.10}$$

The interpolation of the difference between $x_1(t)$ and $\mathbb{E}(x_1(t))$ at the end time of the step has been added into the integrand. Numerical experiments (Section 2.4) demonstrate that (2.10) gives a much better approximation of the variance $\mathrm{Var}(r_4)$.

Armed with the strategy of using the time dependent solution for one species, we can move on to the more general case where we use the time dependent solution of several species.

### 2.3.2 Using the time dependent solution of several species

In many cases there are several species that are limiting the stepsize. They may be linked with each other via the reactions in which they participate. Consider, for example, the motif shown in Figure 2.2. We will refer to this motif as Motif II in the following sections.

A popular model that uses this motif is the enzyme substrate system,

$$E + S \underset{c_2}{\overset{c_1}{\rightleftharpoons}} ES \overset{c_3}{\to} E + P,$$

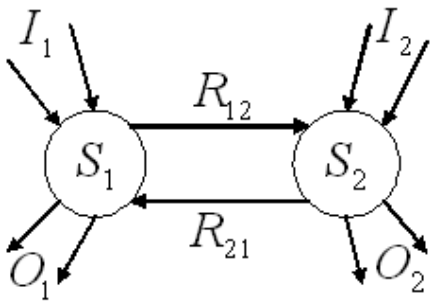



Figure 2.2: Motif II, $I_i$ denotes the set of reactions that generate $S_i$ without consuming $S_j$; $O_i$ denotes the set of reactions that consume $S_i$ without generating $S_j$; $R_{ij}$ denotes the set of reactions that consume $S_i$ and generate $S_j$ at the same time, $i, j = 1, 2,\ i \neq j$.

where $S$ has a huge population while $E$ and $ES$ are present in small populations. Let $\tau_E$, $\tau_S$ and $\tau_{ES}$ denote the stepsizes for $E$, $S$ and $ES$ given by (2.3). It is obvious that $\tau_E,\ \tau_{ES} \ll \tau_S$. Thus if we want to accelerate the simulation, we need to use the time dependent solution for both $E$ and $ES$.

In general, the population of the enzyme is dynamic rather than constant. It can be produced and consumed by other reactions. For example, consider adding the following set of reactions into the enzyme substrate system:

$$R_4 :\ \phi \xrightarrow{a_4} E, \quad R_5 :\ E \xrightarrow{c_5} \phi, \quad R_6 :\ ES \xrightarrow{c_6} \phi.$$

This model is still within the scope of Motif II (see Figure 2.3). The good news is that we have the analytical solution for the time dependent solution of $E$ and $ES$ for the previous system during a stepsize of $\tau_S$ (which implies that $S$ can be treated as constant).

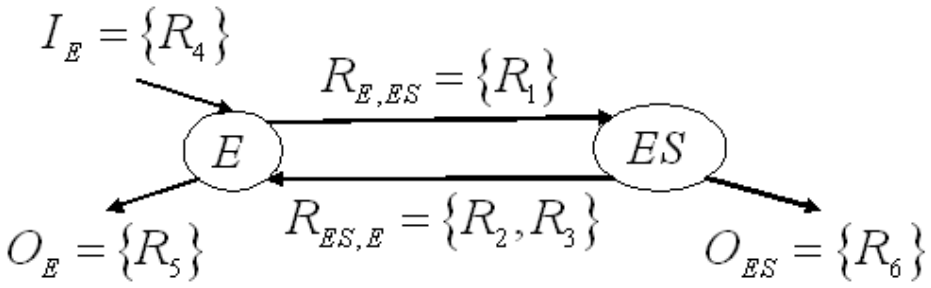



Figure 2.3: E and ES are within the scope of Motif II, $R_4$ is the input reaction for $E$, and $R_5$ and $R_6$ are the output reactions for $E$ and $ES$ respectively. $R_1$ converts $E$ to $ES$, $R_2$ and $R_3$ convert $ES$ to $E$.

Before giving the formula, we define some notation. Let $I_E = \{R_4\}$ be the set of reactions that generate $E$ while not consuming $ES$, $O_E = \{R_5\}$ be the set of reactions that consume $E$ while not producing $ES$, $O_{ES} = \{R_6\}$ be the set of reactions that consume $ES$ while not producing $E$, $R_{E,ES} = \{R_1\}$ be the set of reactions that consume $E$ and generate $ES$, and $R_{ES,E} = \{R_2,\ R_3\}$ be the set of reactions that consume $ES$ and generate $E$.

Similar to the previous example, we have

$$a_I^E = \sum_{R_i \in I_E} a_i = a_4,\ r_I^E = \sum_{R_i \in I_E} r_i = r_4,\ c_{E,ES} = \sum_{R_i \in R_{E,ES}} \tilde{c_i} = c_1 x_S$$

$$c_{ES,E} = \sum_{R_i \in R_{ES,E}} \tilde{c_i} = c_2 + c_3,\ c_O^E = \sum_{R_i \in O_E} \tilde{c_i} = c_5,\ \ c_O^{ES} = \sum_{R_i \in O_{ES}} \tilde{c_i} = c_6 \qquad (2.11)$$

and

$$r_O^E = \sum_{R_i \in O_E} r_i = r_5,\ \ r_O^{ES} = \sum_{R_i \in O_{ES}} r_i = r_6. \qquad (2.12)$$

Here $r_O^E$ and $r_O^{ES}$ are the total number of firings for reactions in $O_E$ and $O_{ES}$.

Using the notation above, the time dependent solution of this system can be written as

$$\begin{aligned}\left(x_E(t),\ x_{ES}(t),\ r_O^E(t),\ r_O^{ES}(t)\right) \sim\ & \mathcal{M}\left(x_E\left(0\right),\ p_1^E(t),\ p_2^E(t),\ p_{O1}^E(t),\ p_{O2}^E(t)\right) \\ & + \mathcal{M}\left(x_{ES}\left(0\right),\ p_1^{ES}(t),\ p_2^{ES}(t),\ p_{O1}^{ES}(t),\ p_{O2}^{ES}(t)\right) \\ & + \mathcal{M}\left(r_I^E, \frac{\lambda_1(t)}{a_I^E t},\ \frac{\lambda_2(t)}{a_I^E t}\ \frac{\lambda_{O1}(t)}{a_I^E t}\ \frac{\lambda_{O2}(t)}{a_I^E t}\right), \end{aligned} \tag{2.13}$$

where the formulas for each parameter are given in Appendix A.1 (see (A.1.28) in Appendix A.1).

This result can be extended from two species to $n$ species $\hat{S} = \{S_1, \ldots, S_n\}$ when the following condition holds:

Condition $(*)$: *For any reaction $R$ that can change the population of a species in $\hat{S}$, one firing of $R$ consumes at most one molecule in $\hat{S}$, and produces at most one molecule in $\hat{S}$.*

A diagram of this general motif is given in Figure 2.4.

Now the definitions in (2.11) and (2.12) can be extended for any $1 \leq i \neq j \leq n$ as follows:

$$a_I^i \triangleq \sum_{R_k \in I_i} a_k,\ r_I^i \triangleq \sum_{R_k \in I_i} r_k,\ c_{ij} \triangleq \sum_{R_k \in R_{ij}} \tilde{c}_k,\ c_O^i \triangleq \sum_{R_k \in O_i} \tilde{c}_k,\ r_O^i \triangleq \sum_{R_k \in O_i} r_k.$$

The time dependent solution for this general motif is given by

$$\left(\boldsymbol{x}\left(t\right), \boldsymbol{r}_O\left(t\right)\right) \sim \sum_{i=1}^{n} \mathcal{M}\left(x_i\left(0\right),\ \boldsymbol{p}^i\left(t\right),\ \boldsymbol{p}_O^i\left(t\right)\right) + \sum_{i=1}^{n} \mathcal{M}\left(r_I^i,\ \frac{1}{a_I^i t}\boldsymbol{\lambda}^i,\ \frac{1}{a_I^i t}\boldsymbol{\lambda}_O^i\right). \tag{2.14}$$

where the formulas for each parameter are given in Appendix A.1. Now that we have the time dependent solution for our motifs, it is time to outline the steps of employing

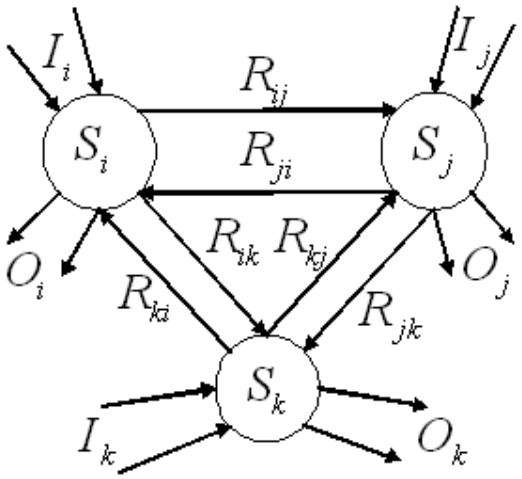



Figure 2.4: General motif

the time dependent solution in tau-leaping, using the enzyme substrate (E-S) system as an example.

1. Use (2.3) to compute the acceptable stepsizes $\tau_i$ for every species (in the E-S example we compute the stepsizes for $E$, $S$ and $ES$). For demonstration purposes, we assume $\tau_1 \leq \tau_2 \leq \cdots \leq \tau_N$ (and in the E-S example we have $\tau_E,\ \tau_{ES} < \tau_S$).

2. Construct the set of species $U$ for which we will use the time dependent solution. Start from the species with the smallest stepsize, i.e. $S_1$. If $S_1$ satisfies condition $(*)$, add it into $U$ to obtain $U = \{\{S_1\}\}$. Now go on to the species which has the second smallest stepsize, i.e. $S_2$. If $\{S_1, S_2\}$ does not satisfy condition $(*)$, end step 2 with $U = \{\{S_1\}\}$. Otherwise, add $S_2$ into $U$. If $S_2$ is linked to $S_1$, i.e. $c_{12} \neq 0$ or $c_{21} \neq 0$, add $S_2$ into $U$ to obtain $U = \{\{S_1,\ S_2\}\}$. Otherwise add it into $U$ to obtain $U = \{\{S_1\}, \{S_2\}\}$. Continue adding species into $U$ in a similar

way until you cannot add any more species that satisfy the condition $(*)$. Now each element in $U$ is a set of species for which we can use the time dependent solution. (In the E-S example we end up with $U = \{\{E,\ ES\}\}$. We cannot add $S$ into $U$ since $\hat{S} = \{E,\ ES,\ S\}$ does not satisfy condition $(*)$, as $R_1$ consumes two molecules in $\hat{S}$).

3. Recompute the stepsize. For species not in $U$, we need to recompute their stepsizes with the new value of each $r_i$ which may no longer be the original Poisson random variable (see Appendix A.3 for a more detailed computation. In the E-S example, we need to recompute the stepsize $\tau_S$).

4. Sample the number of firings for all reactions that do not involve the species in $U$ as reactants. For these reactions tau-leaping is appropriate, so sample Poisson random numbers for them (in the E-S example, $r_4$ is sampled).

5. Sample each element in $U$ using its time dependent solution (2.14). (In the E-S example, $x_E(t)$, $x_{ES}(t)$, $r_O^E(t)$, $r_O^{ES}(t)$ are sampled)

6. For each species $S_i$ in $U$, sample reactions in $O_i$ using the multinomial distribution

$$\{r_j :\ R_j \in O_i\} \sim \mathcal{M}\left(r_O^i,\ \frac{\tilde{c}_j}{c_O^i} :\ R_j \in O_i\right).$$

(In the E-S example, $r_5$ and $r_6$ are sampled, and the multinomial distribution yields $r_5 = r_O^E$, $r_6 = r_O^{ES}$).

7. Sample the reactions in $R_{ij}$. This is not trivial since we have to maintain the flow conservation of the network, so what we actually sample is an instance of a feasible

flow. An algorithm to sample the flow is presented in Appendix A.4. For the E-S example, this step is very simple. First sample $r_1$ using formula (2.10). Here $\mathbb{E}\left(x_E(t)\right)$ in the formula has the form (see Appendix A.1 for detailed derivation)

$$\mathbb{E}\left(x_E(t)\right) = x_E(0)p_1^E(t) + x_{ES}(0)p_1^{ES}(t) + \lambda_1(t),$$

where $p_1^E(t)$, $p_1^{ES}(t)$ and $\lambda_1(t)$ are the parameters that appeared in (2.13).

The conservation equation

$$r_4 + x_E(0) + (r_2 + r_3) = x_E(t) + r_1 + r_5$$

gives

$$(r_2 + r_3) = x_E(t) + r_1 + r_5 - r_4 - x_E(0).$$

Then sample $r_2$ and $r_3$ from their sum using the binomial distribution

$$r_2 = \mathcal{B}\left(x_E(t) + r_1 + r_5 - r_4 - x_E(0),\ \frac{c_2}{c_2 + c_3}\right)$$

$$r_3 = x_E(t) + r_1 + r_5 - r_4 - x_E(0) - r_2.$$

8. If there are reactions involving species in $U$ that are acting as a catalyst, for example

$$S_i \to S_i + S_j,$$

where $S_j$ is not in $U$ (this is guaranteed by the algorithm, because species in $U$ satisfy condition $(*)$), use formula (2.10) to approximate the number of their firings. In the E-S example there is no such reaction.

9. Update the system and begin the next step, or terminate if the end time of the simulation has been reached.

This algorithm is adaptive in the sense that it always applies the time dependent solution to the motifs which limit the tau-leaping stepsize, even though the limiting motifs change during the simulation. We achieve this goal by constructing the limiting motifs $U$ on the fly in step 2, rather than partitioning the system at the beginning of the simulation.

In the enzyme substrate example, allowing non-catalytic bimolecular reactions to be grouped into the motif plays an important role. If such an operation is not allowed, reaction $R_1 : E+S \rightarrow ES$ will be taken away from the motif and we will have a partition of the system as $I_1 = \{R_1\}$, $I_2 = \{R_2, \ldots, R_6\}$. This partition will significantly decrease the stepsize because $I_1$ takes into account only the reaction that converts $E$ to $ES$, while $I_2$ includes the reactions in the opposite direction. Thus if we use a big stepsize, $E$ will be depleted in subsystem $I_1$ in a short time, as will $ES$ in $R_2$. During the remaining time of the step, the system will do nothing. This is obviously not the correct physics of the model. Our method can avoid this partition because we allow $R_1$ to be included in the motif as shown in Figure 2.3. Thus the motif contains all the reactions in both directions and it can take a much longer stepsize than the previous partition.

Table 2.1: The time used for 100000 realizations of the one second simulation for Example 1, $\epsilon = 0.003$

| Method | SSA | Tau Leaping | Tau Leaping/TDS[1] | Tau Leaping/TDS[2] |
|---|---|---|---|---|
| Time used | 5943.97s | 1006.84s | 8.18854s | 1.30296s |

[1]Tau Leaping using time dependent solution of Motif I
[2]Tau Leaping using time dependent solution of Motif II

## 2.4 Numerical simulation

In this section we present the results for the numerical simulations of the examples in Section 2.3. We also demonstrate the time dependent solution for a more complex real world model of blood coagulation.

### 2.4.1 Example 1

The first example is the one mentioned in Section 2.3.1:

$$S_0 \underset{c_2}{\overset{c_1}{\rightleftharpoons}} S_1 \xrightarrow{c_3} S_2, \quad S_1 \xrightarrow{c_4} S_1 + S_3.$$

The parameters are taken to be $c_1 = 0.1,\ c_2 = 1,\ c_3 = 1,\ c_4 = 1$. The initial population of each species is given by $x_0 = 1e + 6,\ x_1 = x_2 = x_3 = 0$. The result of a one second simulation is shown in Table 2.1.

In this example, the stepsize for $S_1$ is smaller than the stepsize for $S_0$, thus the stepsize of tau-leaping is constrained by the stepsize for $S_1$. Using the time dependent solution of $S_1$, we can remove the stepsize requirement of $S_1$ (which tries to keep $x_1$ almost constant during the step) and use the stepsize of $S_0$ for the simulation, which yields a huge speedup. If we use the time dependent solution of both $S_1$ and $S_0$, we

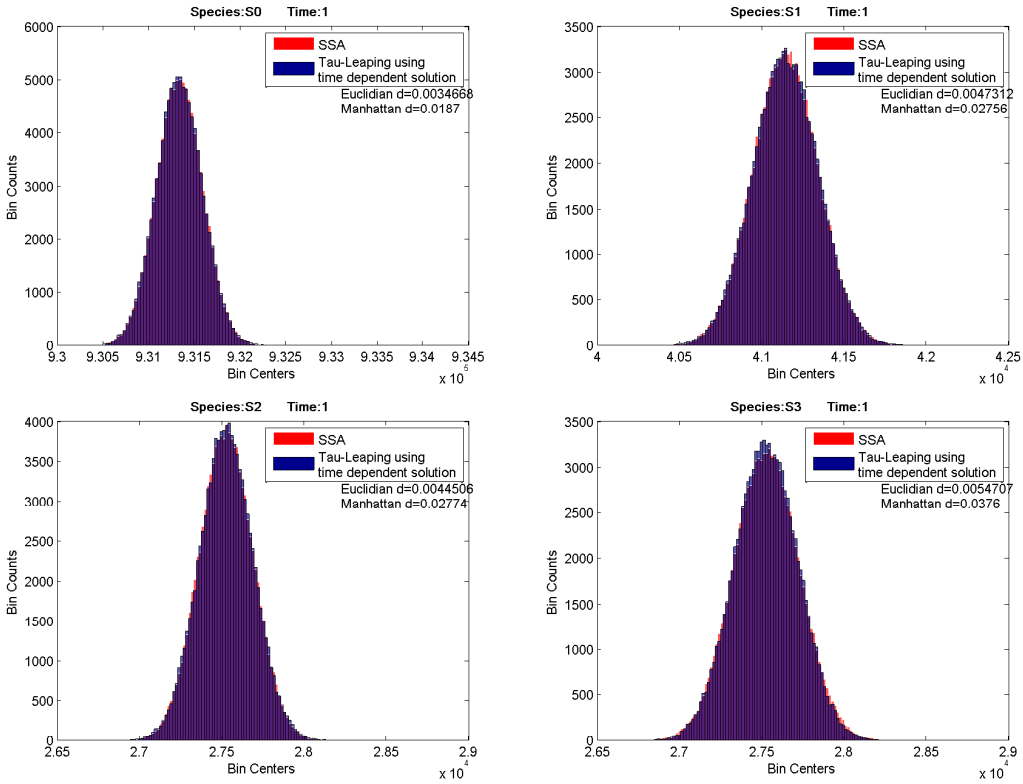



Figure 2.5: Histograms of each species in Example 1. Comparison of result given by SSA and tau-leaping using time dependent solution of Motif II. Red is SSA, blue is tau-leaping using time dependent solution, and purple is the overlap of the two histograms.

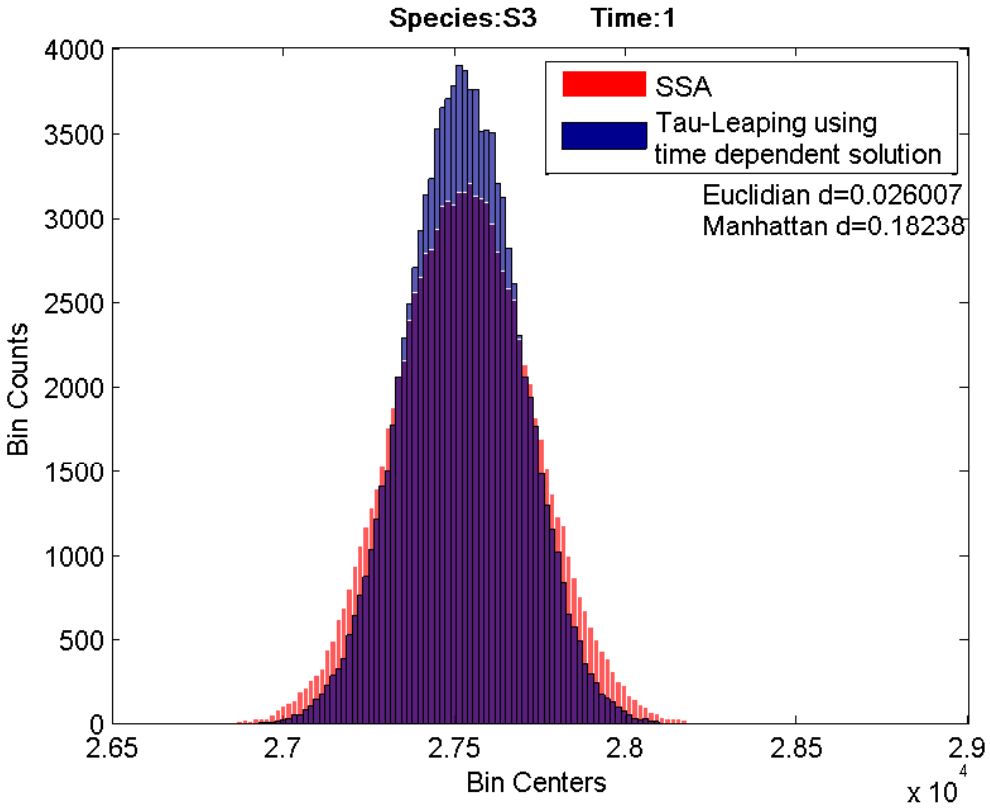



Figure 2.6: The distribution of $S_3$ if (2.9) is used. It has the correct mean value but the variance is too small.

have no stepsize requirement at all! The last method in Table 2.1 simply samples the population of each species at time $t = 1$ directly. This explains why it is so fast.

Speed is important, however we don't want to trade speed at the cost of losing too much accuracy. The population distributions given by SSA and the last method in Table 2.1 are compared in Figure 2.5. The result shows that accuracy is not sacrificed. The distribution of every species is maintained.

Formula (2.10) plays an important role for sampling the population of $S_3$. If we use only the mean value of $x_1$ to do the sampling, i.e. using (2.9), the distribution will have a noticeable error. Figure 2.6 shows the distribution of $S_3$ if (2.9) is used. The distribution has the correct mean but the variance is too small.

### 2.4.2 Example 2

The second example is the one we used in Section 2.3.2:

$$E + S \underset{c_2}{\overset{c_1}{\rightleftharpoons}} ES \xrightarrow{c_3} E + P, \quad \phi \xrightarrow{a_4} E, \quad E \xrightarrow{c_5} \phi, \quad ES \xrightarrow{c_6} \phi.$$

The parameters were taken to be $c_1 = 0.0001,\ c_2 = 10,\ c_3 = c_5 = c_6 = 1,\ a_4 = 100$.The initial population was taken as $x_S = 1e + 6,\ x_E = 1000,\ x_{ES} = x_P = 0$.We do a one second simulation. The results are shown in Table 2.2 and Figure 2.7.

In this example it will not help much if we use the time dependent solution of only one species (the third method in Table 2.2). This is because both $E$ and $ES$ require a small stepsize, thus relaxing the stepsize requirement for one of them will not completely solve our problem. The last method in Table 2.2 uses the time dependent

Table 2.2: The time used for 100000 realizations of a one second simulation of Example 2 with $\epsilon = 0.003$

| Method | SSA | Tau Leaping | Tau Leaping/TDS[1] | Tau Leaping/TDS[2] |
|---|---|---|---|---|
| Time used | 519.708s | 787.655s | 475.314s | 2.57195s |

[1]Tau Leaping using time dependent solution of Motif I
[2]Tau Leaping using time dependent solution of Motif II

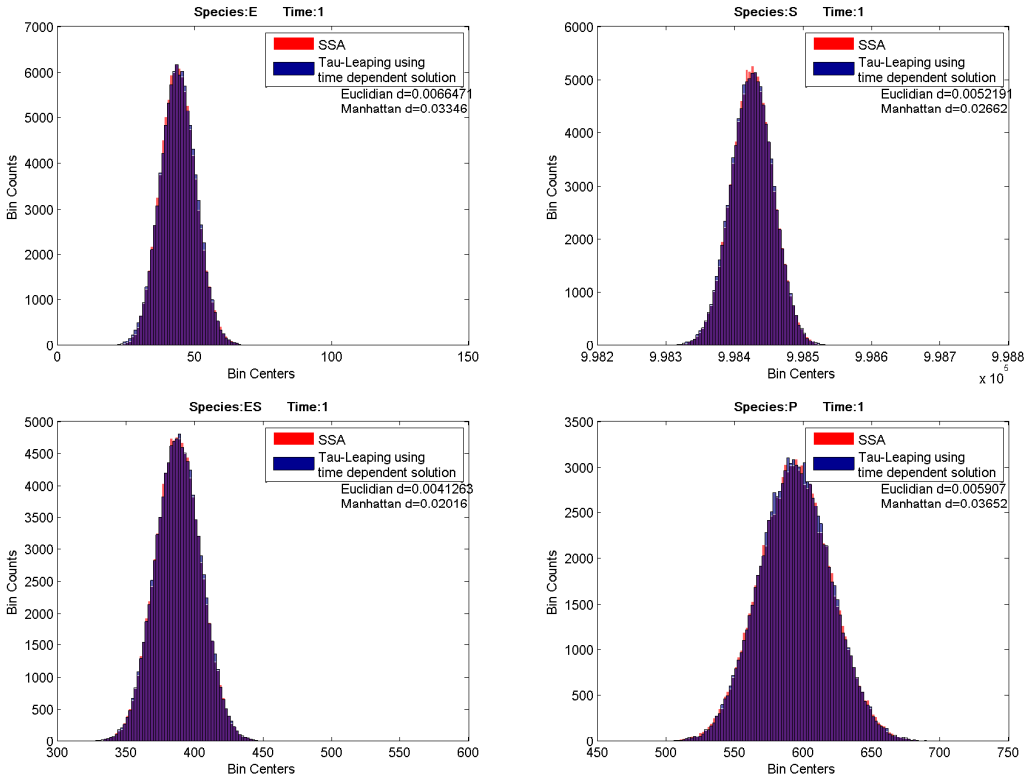



Figure 2.7: Histograms of each species in Example 2. Comparison of result given by SSA and tau-leaping using time dependent solution of Motif II. Red is SSA, blue is tau-leaping using time dependent solution, and purple is the overlap of the two histograms.

Table 2.3: The time used for one realization of a 700-second simulation of the coagulation model, with $\epsilon = 0.02$. The results are averaged over ten realizations.

| Method | SSA | Tau Leaping | Tau Leaping/TDS[1] |
|---|---|---|---|
| Time used | 273.498s | 39.2127s | 7.61337s |

[1]Tau leaping using time dependent solution of Motif I+II.

solution of both $E$ and $ES$, thus the stepsize of the method is actually the stepsize of $S$, which is much larger than those of $E$ and $ES$. In the simulation, the stepsize of $S$ is greater than one second therefore the last method basically samples the population of each species at $t = 1$ directly.

### 2.4.3 Coagulation model

For the final example, we apply our method to a model of blood coagulation [30] with 43 reactions and 33 species. The coagulation model contains reaction pathways that form several levels of cascades. Different factors are activated at different time intervals, which finally leads to the activation of thrombin. Meanwhile, the negative regulation factor antithrombin III binds to thrombin as well as to some other factors in order to control the coagulation process. In this model the species which constrain the stepsize vary as time goes on. However, we do not need to worry about this in the simulation. Our algorithm does not require any prior knowledge about the system. It automatically detects the motifs that limit the stepsize and applies the time dependent solution to them if applicable.

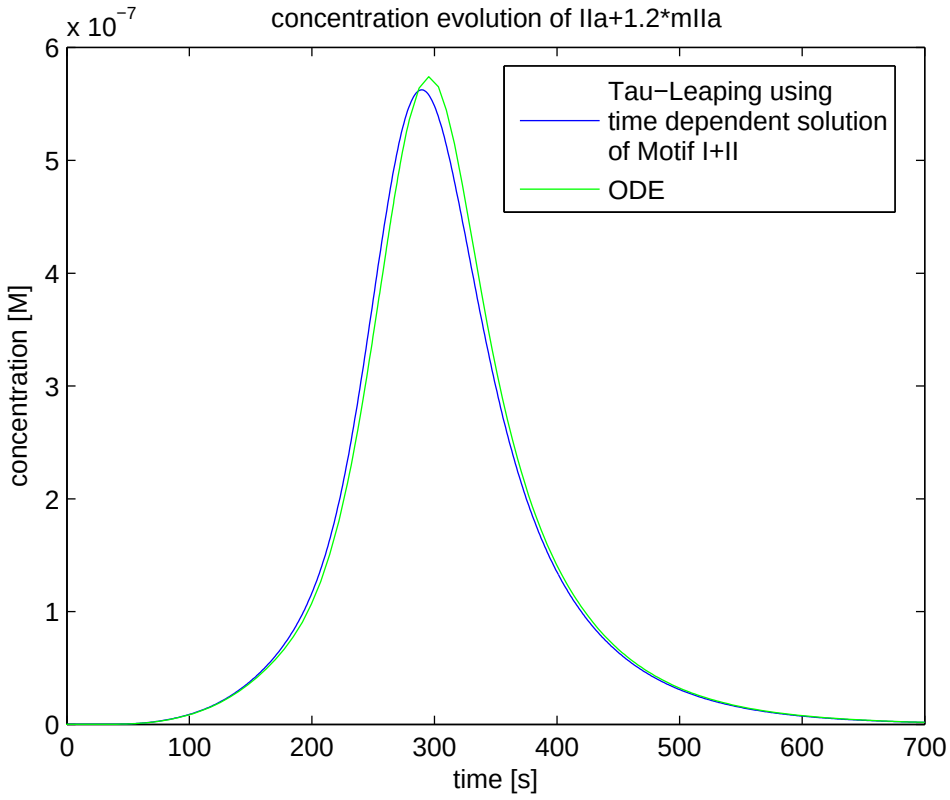



Figure 2.8: Concentration of thrombin (IIa+1.2×mIIa). Blue curve: Tau-leaping using time dependent solution of Motif I+II. Green curve: ODE.

The original model uses concentration for each species rather than population. We convert the concentration to population by selecting a 1mm long cylinder with diameter 0.01mm as the control volume. The time used for one realization of a 700 second simulation is shown in Table 2.3.

The last method in Table 2.3 applies the time dependent solution of Motif I and Motif II. We can see that it already is significantly faster compared to standard tau-leaping. We can expect that if we fully implement the algorithm and use the time dependent solution of motifs containing more than two species, it will further accelerate the speed of the simulation.

According to Table 2.3, if we do a 10000-realization simulation, it takes about 31.7 days for SSA, 4.5 days for tau-leaping, and about 21.1 hours for the time dependent solution implemented as described above. We have code that can run the simulation in parallel. Thus the 10000-realization simulation using the third method required only 5.2 hours running on a 4-core workstation. Since it takes too much time to obtain a complete SSA result of 10000 runs, we do not compare the species distributions for this model. Instead, we compare the evolution of thrombin's mean value with the result given by the ODE model. Here we plot the mean values of IIa+1.2×mIIa given by 10000 tau-leaping runs using the time dependent solution (blue) and the ODE model (green) in Figure 2.8. The error tolerance of the adaptive tau leaping simulation is 0.02, which is larger than the previous examples, so the result will not be as accurate. However Figure 2.8 shows that this result is already able to catch the trend of thrombin.

## 2.5 Conclusion

Tau-leaping using the time dependent solution provides a means to accelerate the simulation of systems that have rapidly changing species. The key point of the method is that it uses the time dependent solution for the fast changing species. Thus, it can use a much larger stepsize than standard tau-leaping, without noticeable loss of accuracy. The auto detection feature grants the algorithm the ability to handle systems whose fast changing species vary over time. However, the method still has some limitations.

1. It can handle only networks that satisfy condition (*). If (*) is violated, we may not have the formula for the time dependent solution. Actually, it is still possible to derive PDEs for the generating function, as we do in Appendix A.1. However the PDEs will be second order and the analytical solution may not be easy to obtain. Even if we find the solution for the PDEs, we still need to convert the generating functions into proper random variables that are easy to sample, which is also nontrivial.

2. For systems that do not have fast-changing species, the method will not benefit the simulation.

The time-dependent solution for acceleration of tau-leaping is already applicable to many real-world systems. The formulas and hence the implementation are

complicated, but we have automated the method so that this is not a limitation. We have implemented the time-dependent solution into the Stochkit 2 [19] software package.

# Chapter 3

# The Time Dependent Propensity Function for Acceleration of Spatial Stochastic Simulation of Reaction-Diffusion Systems

## 3.1 Introduction

The NSM [21] is an efficient algorithm for the simulation of the reaction-diffusion master equation. Approximation-based methods have also been developed for further speeding up the simulation, such as MSA [24] and DFSP [25], which decouple the diffusion and reaction processes. When these approximation-based methods simulate

the reaction process, they freeze the diffusion process. This is an approximation because molecules will be diffusing during that time. In this chapter we present a method that uses the *time dependent propensity function* [27] to sample the reaction events. The time dependent propensity function takes into account the change of the propensity values during a stepsize due to the diffusion process. Thus the method yields a speedup by avoiding the effort of tracking individual diffusion events, while still enjoying excellent accuracy.

This chapter is organized as follows. In Section 3.2 we provide a brief introduction to the SSA. In Section 3.3 we present the new algorithm using the time dependent propensity function. A simple example is used to illustrate the key ideas. Numerical experiments are given in Section 3.4, including application of the method to a realistic model of blood coagulation, and the algorithm is briefly summarized in Section 3.5. Detailed mathematical derivations are provided in the Appendix of this thesis. The work described in this chapter was published in *The time dependent propensity function for acceleration of spatial stochastic simulation of reaction-diffusion systems* (Jin Fu, Sheng Wu, Hong Li, and Linda R. Petzold. J. Comput. Phys., 274:524–549, 2014).

## 3.2 Stochastic simulation algorithm

Consider a homogeneous system of $N$ species $S_1, \ldots, S_N$ and $M$ reactions $R_1, \ldots, R_M$. The state vector of the system is denoted by $\boldsymbol{X} = \{x_1, \ldots, x_N\}$, where $x_i$ is the population of species $i$. The SSA is based on the well-mixed assumption. The

probability that reaction $R_i$ fires in an infinitesimal interval $dt$ is given by $a_i(\boldsymbol{X})dt$, where $a_i(\boldsymbol{X})$ is the propensity function of $R_i$. In every step, the algorithm advances the system by sampling the time to the next reaction and the reaction that will fire. Finally it updates the state of the system.

To sample the next reaction time, the SSA uses the total propensity $a_0(\boldsymbol{X}) = \sum_{i=1}^{M} a_i(\boldsymbol{X})$ of the system. As the probability that the system will fire a reaction in the next infinitesimal $dt$ is $a_0(\boldsymbol{X})\,dt$, the time to the next reaction follows an exponential distribution with parameter $a_0(\boldsymbol{X})$. This is the distribution that the SSA uses to sample the next reaction time.

To sample the reaction that the system should fire, the SSA selects the next reaction with probability proportional to its propensity. Thus the probability of choosing reaction $i$ is $a_i(\boldsymbol{X})/a_0(\boldsymbol{X})$. Finally, the SSA updates the system state and repeats these steps until the simulation is completed.

## 3.3 Spatial stochastic simulation using the time dependent propensity function (TDPD)

The SSA performs two tasks in each step: select the time to the next reaction and select the reaction to be fired. Analogously, TDPD divides each step of the spatial stochastic simulation into the two tasks described above. In this section we illustrate how these two tasks are performed in TDPD, using the following simple example. In

the spatial stochastic simulation, the state $\boldsymbol{X}$ is given by the number of molecules of each species in each voxel.

The example system is composed of two voxels and a reaction $A + B \xrightarrow{c} C$, where $c$ is the rate constant of the reaction. An $A$ molecule and a $B$ molecule are able to react only when they are in the same voxel. Molecules $A$, $B$ and $C$ can jump between the two voxels with diffusion propensities $\kappa^A$, $\kappa^B$, $\kappa^C$ respectively. Initially, there are $X_1^A$ $A$ molecules in voxel 1 and $X_2^B$ $B$ molecules in voxel 2.

The first step of our algorithm is to select the next reaction time.

### 3.3.1 Select the time to the next reaction using the time dependent propensity

In this section we show how to sample the next reaction time. To achieve this goal, we must find the distribution of next reaction times. This distribution depends on the propensity function, which is a function of time.

#### 3.3.1.1 The distribution of next reaction times for TDPD

This section basically restates the procedure that SSA uses to obtain the distribution of the next reaction time, but in a spatial setting. The conclusion in this subsection also appeared in [21] and [28] (which can trace back to [31]), where the time dependent propensity is applied for simulation algorithms in different scenarios.

Let $\mathbf{X}_0$ be the initial state of the system and $a_0(t, \mathbf{X}_0)$ the total propensity of the system at time $t$ under the condition that no reaction occurs before $t$. Then the

probability $Q(t, X_0)$ that no reaction occurs before $t$ satisfies

$$Q(t + dt, \mathbf{X}_0) = Q(t, \mathbf{X}_0)(1 - a_0(t, \mathbf{X}_0)\, dt),$$

which yields the ODE

$$\frac{dQ(t, \mathbf{X}_0)}{dt} = -Q(t, \mathbf{X}_0)\, a_0(t, \mathbf{X}_0),$$

whose solution is given by

$$Q(t, \mathbf{X}_0) = e^{-\int_0^t a_0(s, \mathbf{X}_0) ds} \Rightarrow P(t, \mathbf{X}_0) \triangleq 1 - Q(t, \mathbf{X}_0) = 1 - e^{-\int_0^t a_0(s, \mathbf{X}_0) ds}, \quad (3.1)$$

where $P(t, X_0)$ is the probability that the next reaction occurs before time $t$.

In the SSA, $a_0(t, \mathbf{X}_0)$ is a constant before the next reaction. However, in the spatial case it is a function of time $t$, because the diffusion process changes the system state over time. Similar to sampling the next reaction time in SSA, the time to the next reaction can be obtained by solving

$$\hat{r} = P(t, \mathbf{X}_0), \quad (3.2)$$

where $\hat{r}$ is a uniformly distributed random number in $(0, 1)$. Using (3.1) and (3.2) yields

$$-\ln(1 - \hat{r}) = \int_0^t a_0(s, \mathbf{X}_0)\, ds.$$

Since $r \triangleq 1 - \hat{r}$ is also a uniform random number in $(0, 1)$, it is equivalent to restate the above as

$$-\ln r = \int_0^t a_0(s, \mathbf{X}_0)\, ds. \quad (3.3)$$

Next, we must find $a_0(t, \mathbf{X}_0)$.

#### 3.3.1.2 The time dependent propensity function

As stated above, $a_0(t, \mathbf{X}_0)\,dt$ is the probability that a reaction will fire in the time interval $[t, t+dt]$, given that no reaction fires before $t$. This probability is the sum of the probabilities of every possible reaction event during $[t, t+dt]$. Let us look at a particular $A$ molecule in voxel 1 and a $B$ molecule in voxel 2 in our example. Under the condition that no reaction fires before time $t$, the probability that they will react during $[t, t+dt]$ is

$$\begin{aligned}
&P(\text{the two molecules react in } [t, t+dt]) \\
=&P(\text{they are in voxel 1 at time } t \text{ and then react in } [t, t+dt]) \\
&+ P(\text{they are in voxel 2 at time } t \text{ and then react in } [t, t+dt]) \\
=&P(\text{they are in voxel 1 at time } t) \times cdt + P(\text{they are in voxel 2 at time } t) \times cdt,
\end{aligned} \tag{3.4}$$

where $c$ is the reaction rate.

The probability terms in (3.4) are not trivial. However if we take the assumption that the system is undergoing a pure diffusion process between the reaction events, it simplifies the problem. Under this assumption, molecules diffuse independently and their location distribution is the solution of the master equation of the diffusion process.

Thus

$$
\begin{aligned}
&P\left(\text{the two molecules are in voxel 1 at time } t\right) \\
=&P\left(\text{A remains in voxel 1 at time } t\right) \times P\left(\text{B diffuses from voxel 2 to voxel 1 by time } t\right) \\
\triangleq&p_{11}^{A}\left(t\right) p_{21}^{B}\left(t\right),
\end{aligned}
$$

where $p_{ij}^{k}(t)\,(i,j=1,2;k=A,B)$ is the probability that the molecule of species $k$ diffuses from voxel $i$ to voxel $j$ by time $t$.

Now, under the condition that no reaction occurs before $t$, (3.4) can be written as

$$
P\left(\text{the two molecules react in } \left[t,t+dt\right]\right)=p_{11}^{A}\left(t\right)p_{21}^{B}\left(t\right)cdt+p_{12}^{A}\left(t\right)p_{22}^{B}\left(t\right)cdt. \quad (3.5)
$$

Another benefit of assuming that the system is governed by a diffusion process between the reaction events is that the master equation of the discrete one dimensional diffusion process with finite voxels and reflecting boundary conditions has a closed form solution (see Appendix A.5), which also serves as the foundation for constructing the solutions of higher dimensional diffusion processes. In the example case of two voxels, $p_{ij}^{k}\left(t\right)\left(k=A,B\right)$ is given by

$$
\begin{pmatrix} p_{11}^{k} & p_{12}^{k} \\ p_{21}^{k} & p_{22}^{k} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1+e^{-2\kappa^{k}t} & 1-e^{-2\kappa^{k}t} \\ 1-e^{-2\kappa^{k}t} & 1+e^{-2\kappa^{k}t} \end{pmatrix}, \quad (k=A,B), \qquad (3.6)
$$

where $\kappa^{k}$ is the diffusion propensity for species $k$.

Inserting (3.6) into (3.5) yields the probability for a particular pair of molecules to react during $[t,t+dt]$. Since there are $X_1^A \times X_2^B$ such pairs, the total probability of

such events is

$$P\left(\text{a reaction occurs in } [t, t+dt] \text{ given that no reaction occurs before } t\right)$$

$$= X_1^A X_2^B \left(p_{11}^A(t)\, p_{21}^B(t)\, cdt + p_{12}^A(t)\, p_{22}^B(t)\, cdt\right) = a_0(t, \mathbf{X}_0)\, dt.$$

Thus,

$$a_0(t, \mathbf{X}_0) = cX_1^A X_2^B \left(p_{11}^A(t)\, p_{21}^B(t) + p_{12}^A(t)\, p_{22}^B(t)\right) = \frac{c}{2} X_1^A X_2^B \left(1 - e^{-2\left(\kappa^A + \kappa^B\right)t}\right). \tag{3.7}$$

Inserting (3.7) into (3.3) yields the formula for sampling the next reaction time,

$$-\ln r = \int_0^t a_0(s, \mathbf{X}_0)\, ds = \frac{c}{2} X_1^A X_2^B \left(t + \frac{e^{-2\left(\kappa^A + \kappa^B\right)t} - 1}{2\left(\kappa^A + \kappa^B\right)}\right). \tag{3.8}$$

Here we note that (3.6), which results from the assumption that the system is undergoing a diffusion process with reflecting boundary conditions, is only an approximation to the true spatial distribution. To make this point clearer, we denote the true value of $p_{ij}^k$ by $\tilde{p}_{ij}^k$ $(k = A, B)$ and take a look at what this value is supposed to be.

#### 3.3.1.3 Error analysis of $p_{ij}^k$

Consider a particular $A$ molecule that initially remains in voxel 1. Denote by $\mathcal{R}$ the set of reaction events in which this molecule is involved as a reactant, and by $\overline{\mathcal{R}}$ the set of all other reaction events. At any time $t > 0$, under the condition that no event in $\overline{\mathcal{R}}$ occurs before $t$, there are only three possible states for the observed $A$ molecule: it is in voxel 1, it is in voxel 2, or it is already consumed by a reaction event in $\mathcal{R}$. Denote the probabilities of these three states by $p_1(t)$, $p_2(t)$ and $p_r(t)$ respectively.

By definition, $\tilde{p}_{ij}^{A}(t)$ is the probability that an $A$ molecule diffuses from voxel $i$ to voxel $j$ at time $t$, given that no reaction occurs before $t$. Thus its true value, for example $\tilde{p}_{1j}^{A}(t)\,(j=1,2)$, is given by

$$\begin{aligned}\tilde{p}_{1j}^{A}(t) &= \frac{p_j(t)}{p_1(t)+p_2(t)} = \frac{p_j(t)}{p_1(t)+p_2(t)}\left(p_1(t)+p_2(t)+p_r(t)\right)\\ &= p_j(t) + \frac{p_j(t)}{p_1(t)+p_2(t)}p_r(t) \triangleq p_j(t)+\tilde{r}_j(t), \quad (j=1,2).\end{aligned}$$

Here $\tilde{r}_j(t)$ is defined as $p_r(t)p_j(t)/(p_1(t)+p_2(t))$. It is clear that $\tilde{r}_j(t) \le p_r(t)$, and

$$\tilde{r}_1(t)+\tilde{r}_2(t) = p_r(t). \tag{3.9}$$

Since $p_{1j}^{A}(t)$ is greater than $p_j(t)$ (see Appendix A.7 for the proof), it can also be decomposed into $p_j(t)$ plus some positive value, say $r_j(t)$. Thus the difference between the true value $\tilde{p}_{1j}^{A}(t)$ and its approximation $p_{1j}^{A}(t)$ can be written as

$$\left|p_{1j}^{A}(t)-\tilde{p}_{1j}^{A}(t)\right| = \left|\left(p_j(t)+r_j(t)\right)-\left(p_j(t)+\tilde{r}_j(t)\right)\right| = \left|r_j(t)-\tilde{r}_j(t)\right|.$$

We can use this equation to bound the difference between $\tilde{p}_{1j}^{A}(t)$ and $p_{1j}^{A}(t)$.

A bound for $r_j(t)$ can be obtained from

$$1 = \sum_j p_{1j}^{A}(t) = \sum_j \left(p_j(t)+r_j(t)\right),$$

which implies

$$\sum_j r_j(t) = 1-\sum_j p_j(t) = p_r(t). \tag{3.10}$$

Thus

$$r_j(t) \le p_r(t) \quad (j=1,2),$$

and an upper bound for $\left|p_{1j}^{A}(t)-\tilde{p}_{1j}^{A}(t)\right|$ is given by

$$\left|p_{1j}^{A}(t)-\tilde{p}_{1j}^{A}(t)\right|=\left|r_j(t)-\tilde{r}_j(t)\right|\leq\max\left(r_j(t),\tilde{r}_j(t)\right)\leq p_r(t)\quad(j=1,2).$$

The sum of these differences over all voxels is bounded by

$$\begin{aligned}&\sum_j\left|p_{1j}^{A}(t)-\tilde{p}_{1j}^{A}(t)\right|=\sum_j\left|r_j(t)-\tilde{r}_j(t)\right|\\&\leq\sum_j\left(r_j(t)+\tilde{r}_j(t)\right)=\sum_j r_j(t)+\sum_j\tilde{r}_j(t)=2p_r(t).\end{aligned}$$

Here the last equality arises from equations (3.9) and (3.10). Thus the error in $p_{1j}^{A}(t)$ has an upper bound which is determined by $p_r(t)$. But how large can $p_r(t)$ be during a simulation step?

Since $p_r(t)$ by definition is the probability of the observed $A$ molecule being consumed by a reaction before $t$, given that no reaction events in $\overline{\mathcal{R}}$ occur before $t$, the longer the time, the larger that probability will be. As our simulation step size $\tau$ is the time to the next reaction, $p_r(t)$ in a simulation step will take its maximum value at $t=\tau$. Since $\tau$ is a random variable, $p_r(\tau)$ itself is also a random variable. It can be shown that the expectation of $p_r(\tau)$ has an upper bound given by (see Appendix A.6)

$$\mathrm{E}\left(p_r(\tau)\right)\leq\max_t\frac{a(t)}{a_0(t)+a(t)},\tag{3.11}$$

where $a(t)$ is the propensity contributed by the observed $A$ molecule, which is defined as $a(t)=a_0(t)-a_{\bar{\mathcal{R}}}(t)$ where $a_0(t)$ is the total propensity of the system at time $t$ given that no reaction occurs before $t$, i.e. the total propensity of reaction events in $\mathcal{R}\cup\overline{\mathcal{R}}$ at time $t$ given that no reaction events in $\mathcal{R}\cup\overline{\mathcal{R}}$ occur before $t$. And $a_{\bar{\mathcal{R}}}(t)$

is the total propensity of the reaction events in $\overline{\mathcal{R}}$ at time $t$ given that no event in $\overline{\mathcal{R}}$ occurs before $t$. Intuitively, $a_{\overline{\mathcal{R}}}(t)$ measures the propensity of reaction events where the observed molecule is not involved. Thus $a_0(t) - a_{\overline{\mathcal{R}}}(t)$ can be regarded as the amount of propensity value that contributed by the observed molecule. When there are many $A$ molecules, $\mathrm{E}\left(p_r\left(\tau\right)\right)$ will be small. In this paper we will assume that this condition holds for the systems we consider. i.e. the propensity contributed by a particular molecule is much smaller than the total propensity $a_0$ of the overall system.

Besides the error analysis, the computational cost of solving (3.3) is also important. This topic will be discussed in the next subsection.

#### 3.3.1.4 Complexity of solving (3.3)

In the two-voxel example, the propensity function has the form (3.7), and equation (3.3) leads to the expression in (3.8). In general if we have $L$ voxels, in voxel $i$ $(i = 1, \ldots, L)$ we initially have $X_i^A$ $A$ molecules and $X_i^B$ $B$ molecules. Then the total propensity is given by

$$a_0\left(t, \mathbf{X}_0\right) = c \sum_{i=1}^{L} \sum_{j=1}^{L} X_i^A X_j^B \left( \sum_{k=1}^{L} p_{ik}^A\left(t\right) p_{jk}^B\left(t\right) \right) = c \left(\mathbf{X}^A\right)^T \mathbf{P}^A(t) \left(\mathbf{P}^B(t)\right)^T \mathbf{X}^B, \tag{3.12}$$

where $\mathbf{X}^A$ is the population vector of species $A$ and $\mathbf{P}^A(t)$ is the transition matrix of species $A$, whose element at row $i$ and column $j$ is $p_{ij}^A(t)$, and similarly for species $B$.

From equation (A.5.5) in Appendix A.5, the matrix $\mathbf{P}^A(t)$ has the form

$$\left(\mathbf{P}^A(t)\right)^T = \mathbf{V} \mathrm{diag}\left(e^{\kappa^A \lambda_0 t}, \ldots, e^{\kappa^A \lambda_{L-1} t}\right) \mathbf{V}^{-1} \left(\mathbf{P}^A(0)\right)^T. \tag{3.13}$$

Here $\mathbf{V}$ is the matrix consisting of the eigenvectors of (A.5.4). In the simulation it is convenient to normalize the eigenvectors, so that $\mathbf{V}$ has the properties

$$\mathbf{V}^{-1} = \mathbf{V}^T, \quad \mathbf{V}^T\mathbf{V} = \mathbf{I}. \tag{3.14}$$

$\mathbf{P}^A(0)$ is the initial value of the transition matrix $\mathbf{P}^A(t)$. In a simulation with given initial positions, $\mathbf{P}^A(0) = \mathbf{I}$.

Plugging (3.13) into (3.12), noting properties (3.14) and setting $\mathbf{P}^A(0) = \mathbf{I}$, we obtain

$$\begin{aligned} a_0\left(t, \mathbf{X}_0\right) &= c\left(\mathbf{X}^A\right)^T \mathbf{P}^A(t)\left(\mathbf{P}^B(t)\right)^T \mathbf{X}^B \\ &= c\left(\mathbf{X}^A\right)^T \mathbf{V}\mathrm{diag}\left(e^{\left(\kappa^A+\kappa^B\right)\lambda_0 t}, \ldots, e^{\left(\kappa^A+\kappa^B\right)\lambda_{L-1} t}\right)\mathbf{V}^T\mathbf{X}^B. \end{aligned} \tag{3.15}$$

The integral of $a_0\left(t, \mathbf{X}_0\right)$ can be expressed analytically using (3.15), thus (3.3) becomes, for this example,

$$-\ln r = c\left(\mathbf{V}^T\mathbf{X}^A\right)^T \mathrm{diag}\left(\frac{e^{\left(\kappa^A+\kappa^B\right)\lambda_0 t}-1}{\left(\kappa^A+\kappa^B\right)\lambda_0}, \ldots, \frac{e^{\left(\kappa^A+\kappa^B\right)\lambda_{L-1} t}-1}{\left(\kappa^A+\kappa^B\right)\lambda_{L-1}}\right)\mathbf{V}^T\mathbf{X}^B. \tag{3.16}$$

It is clear now that the right hand side of (3.15) and (3.16) requires: (a) matrix – vector multiplications ($\mathbf{V}^T\mathbf{X}^A$ and $\mathbf{V}^T\mathbf{X}^B$) and (b) vector – diagonal matrix – vector multiplication. For (a), The computational cost is $O\left(L^2\right)$. For (b), the computational cost is $O(L)$. If we have multiple such reaction channels, we need to repeat (a) and (b) multiple times. However the cost for (a) can be reduced if a species is a reactant for several reactions, since we need only to perform the matrix – vector multiplication for this species once and reuse the result whenever needed. During the following Newton

iterations, (a) brings no additional cost, as it needs only to be computed once when the equation is constructed. However, (b) must be recomputed in every iteration.

The computational cost of solving (3.16) does not explicitly depend on the population of each species. Increasing the population of species $A$ only changes the elements of vector $\mathbf{X}^A$, which does not affect the computational complexity. However the more molecules in the system, the more reaction events would occur, thus the more simulation steps are required. Therefore, the molecule population still affects the simulation cost, but not by making (3.16) harder to solve.

It is worth mentioning that the diffusion propensities $\kappa^A$ and $\kappa^B$ do not affect the complexity of (3.16) as well. Unlike the molecule population which may affect the number of reaction events, diffusion propensities affect the number of diffusion events. Since we need only to solve (3.16) for reaction events, diffusion events do not add computational overhead to the simulation. This is an advantage over the algorithms which track diffusion events. It enables us to simulate systems with large diffusion propensities without extra computational effort. In the case of increasing resolution, e.g. divide each voxel into $n$ smaller voxels, the algorithm incurs the overhead due to the increased value of $L$. However, algorithms that track diffusion events incur additional costs due to the large propensities for diffusive transfers. A similar analysis applies to second order reactions like $A + A \to C$.

After settling the problem of selecting the next reaction time, our next task is to select a reaction to fire.

### 3.3.2 Select the next reaction

In the SSA, the probability that a reaction is selected is proportional to its propensity. For the spatial simulation we will use the same idea. Thus we need first to specify the set of all possible reaction events, and then select one from the set.

#### 3.3.2.1 The set of reaction events and their propensities

Since we have already sampled the time $\tau$ to the next reaction, a typical reaction event is that the system diffuses from the initial state $\mathbf{X}_0$ to a new state $\mathbf{Y}$ at time $\tau$ and then fires a reaction in the infinitesimal time interval $[\tau, \tau + dt]$. The probability $p_{\mathbf{Y}}$ of this event is

$$p_{\mathbf{Y}} = P\left(\text{diffuse from } \mathbf{X}_0 \text{ to } \mathbf{Y} \text{ at time } \tau\right) \times P\left(\text{fire a reaction in } [\tau, \tau + dt] \text{ given state } \mathbf{Y}\right). \tag{3.17}$$

Our purpose in this section is to select a possible state $\mathbf{Y}$ at time $\tau$, proportional to the probability $p_{\mathbf{Y}}$, and then select a reaction to fire. It is clear from (3.17) that if a state $\mathbf{Y}$ has no possible reaction to fire, e.g. all $A$ molecules in one voxel and all $B$ molecules in another, then $p_{\mathbf{Y}}$ will be zero and the probability that this state is selected is zero.

#### 3.3.2.2 The sampling algorithm

Directly using (3.17) to do the sampling work is not easy. Here we will sample the reaction from another point of view. We sum up the propensities of all potential reaction events at time $\tau$ and select one according to its propensity. In our example, the

probability of a particular $A$ molecule from voxel $i$ and a particular $B$ molecule from voxel $j$ to diffuse to voxel $k$ at time $\tau$ and then react during $[\tau, \tau + dt]$ is $p_{ik}^A(\tau)p_{jk}^B(\tau)cdt$, so the propensity of this particular reaction event at time $\tau$ is $cp_{ik}^A(\tau)p_{jk}^B(\tau)$. Summing over all such events yields the total propensity

$$a_0(\tau) = \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2} X_i^A X_j^B cp_{ik}^A(\tau)p_{jk}^B(\tau),$$

where $X_i^A$ and $X_j^B$ are the initial populations of $A$ molecules in voxel $i$ and $B$ molecules in voxel $j$. In the SSA, the probability of a reaction to be selected is proportional to its propensity. Here we use the same idea. The probability that we select an event that an $A$ molecule from voxel $i$ and a $B$ molecule from voxel $j$ react in voxel $k$ at time $\tau$ is $X_i^A X_j^B cp_{ik}^A(\tau)p_{jk}^B(\tau) \Big/ a_0(\tau)$.

After sampling the reaction event, it is time for us to update the system. Suppose that the sampled reaction event is that an $A$ molecule from voxel $i_0$ and a $B$ molecule from voxel $j_0$ react in voxel $k_0$ at time $\tau$. As the sampling result by definition specifies the voxel location of the two reactant molecules, there is no need to sample a diffusion process for these two molecules. So we first remove an $A$ molecule from voxel $i_0$ and a $B$ molecule from voxel $j_0$. Then we sample a diffusion process for the remaining system up to time $\tau$. Finally, we insert a product molecule C into voxel $k_0$. This completes the procedure of firing the selected reaction.

Now we have completed a step of the simulation for our simple example. The next subsection summarizes the algorithm.

### 3.3.3 Summary of the algorithm

In this section we present the algorithm in a more general setting. Suppose that a one dimensional system has $M$ reactions, $N$ species and $L$ voxels. Assume the current state of the system is $\mathbf{X}$, and without loss of generality, the current time is 0. Then the time dependent propensity functions for different types of reactions are

- $\phi \xrightarrow{c} something$

$$a\left(t,\mathbf{X}\right)=n\times c$$

  where $n$ is the number of voxels that contain the reaction.

- $A \xrightarrow{c} something$

$$a\left(t,\mathbf{X}\right)=c\sum_{i=1}^{L}X_{i}^{A}$$

- $A+B \xrightarrow{c} something$

$$a\left(t,\mathbf{X}\right)=c\sum_{i,j,k=1}^{L}X_{i}^{A}X_{j}^{B}p_{ik}^{A}\left(t\right)p_{jk}^{B}\left(t\right)$$

- $A+A \xrightarrow{c} something$

$$\begin{aligned}a\left(t,\mathbf{X}\right)&=c\sum_{i<j}\sum_{k=1}^{L}X_{i}^{A}X_{j}^{A}p_{ik}^{A}\left(t\right)p_{jk}^{A}\left(t\right)+\sum_{i,k=1}^{L}\frac{c}{2}X_{i}^{A}\left(X_{i}^{A}-1\right)\left(p_{ik}^{A}\left(t\right)\right)^{2}\\&=\frac{c}{2}\left(\sum_{i,j,k=1}^{L}X_{i}^{A}X_{j}^{A}p_{ik}^{A}\left(t\right)p_{jk}^{A}\left(t\right)-\sum_{i,k=1}^{L}X_{i}^{A}\left(p_{ik}^{A}\left(t\right)\right)^{2}\right),\end{aligned}$$

and the total propensity is given by

$$a_{0}\left(t,\mathbf{X}\right)=\sum_{i=1}^{M}a_{i}\left(t,\mathbf{X}\right),$$

where $a_i(t, \mathbf{X})$ is the propensity function of reaction $i$ at time $t$ given that no reaction occurs before $t$. Here "$\rightarrow$ *something*" could be "$\rightarrow \phi$" which denotes a reaction that only consumes molecules.

The simulation steps of the TDPD algorithm are listed below

0. Compute the eigenvalues and eigenvectors using (A.5.3) and (A.5.4) (These values need only to be computed once).

For each realization, do the following:

1. Initialize the time $t = t_0$ and the system state $\mathbf{X} = \mathbf{X}_0$.

2. With the system in state $\mathbf{X}$ at time $t$, generate a uniform random number $r \sim \mathrm{U}(0, 1)$ and solve the following equation to obtain a sample $\tau$ of the time to the next reaction,
$$-\ln r = \int_0^{\tau} a_0(s, \mathbf{X})\, ds. \tag{3.18}$$

3. Compute the transition matrix $p_{ij}(\tau)$ for each diffusive species using equation (A.5.5).

4. Sample the reaction $R_l$ to fire. Its index $l$ is an integer random variable between 1 to $M$ with point probabilities
$$P(l) = \frac{a_l(\tau, \mathbf{X})}{a_0(\tau, \mathbf{X})}.$$

5. Sample where the reactant molecules come from and where the product is generated.

- If in step 4 the sampled reaction $R_l$ is $\phi \xrightarrow{c} something$, suppose that there are $n$ voxels which contain the reaction, and the reaction occurs in voxel $k$. Then $k$ is a random variable with point probability

$$P(k) = \begin{cases} 1/n & \text{if voxel } k \text{ contains the reaction} \\ 0 & \text{if voxel } k \text{ does not contain the reaction} \end{cases}.$$

- If in step 4 the sampled reaction $R_l$ is $A \xrightarrow{c} something$, suppose that the reactant originates in voxel $i$ and the product is produced in voxel $k$. Then $(i, k)$ is a random variable with point probability (Note that voxel $i$ and voxel $k$ are not necessarily adjacent).

$$P(i,k) = \frac{cX_i^A p_{ik}^A(\tau)}{a_l(\tau, \mathbf{X})}, \quad i,k = 1,\ldots,L.$$

- If in step 4 the sampled reaction $R_l$ is $A + B \xrightarrow{c} something$, supposing that reactant $A$ originates in voxel $i$, reactant $B$ originates in voxel $j$, and the product is produced in voxel $k$, then $(i, j, k)$ is a random variable with point probability

$$P(i,j,k) = \frac{cX_i^A X_j^B p_{ik}^A(\tau)\, p_{jk}^B(\tau)}{a_l(\tau, \mathbf{X})}, \quad i,j,k = 1,\ldots,L.$$

- If in step 4 the sampled reaction $R_l$ is $A+A \xrightarrow{c} something$, supposing that the two molecules originate in voxel $i$ and voxel $j$, and the product is produced in voxel $k$, then without loss of generality, we assume $i \leq j$. $(i, j, k)$ is a

random variable with point probability

$$P(i,j,k) = \begin{cases} \frac{cX_i^A X_j^A p_{ik}^A(\tau) p_{jk}^A(\tau)}{a_l(\tau,\mathbf{X})} & i<j \\ \\ \frac{\frac{c}{2}X_i^A\left(X_i^A-1\right)\left(p_{ik}^A(\tau)\right)^2}{a_l(\tau,\mathbf{X})} & i=j \end{cases}, \quad i,j,k=1,\ldots,L.$$

6. Remove the reactant molecules from the current state $\mathbf{X}$.

   - If in step 4 the sampled reaction $R_l$ is $\phi \xrightarrow{c}$ *something*, skip this step.
   - If in step 4 the sampled reaction $R_l$ is $A \xrightarrow{c}$ *something* and in step 5 the sampled voxel where the reactant originates is $i$, then decrease $X_i^A$ by one.
   - If in step 4 the sampled reaction $R_l$ is $A+B \xrightarrow{c}$ *something* and in step 5 the sampled voxels where the reactants $A$, $B$ originate are $(i,j)$ respectively, then decrease $X_i^A$ and $X_j^B$ by one.
   - If in step 4 the sampled reaction $R_l$ is $A+A \xrightarrow{c}$ *something* and in step 5 the sampled voxels where the two reactants originate are $(i,j)$, decrease $X_i^A$ by one, then decrease $X_j^A$ by one.

7. Sample a diffusion process with reflecting boundary conditions up to time $t+\tau$. For example, for species $A$, sample a multinomial random variable for each voxel $i$ $(i=1,\ldots,L)$,

$$\hat{\mathbf{Y}}_i = \mathcal{M}\left(X_i^A, p_{i1}^A(\tau),\ldots,p_{iL}^A(\tau)\right).$$

   Here $\hat{\mathbf{Y}}_i = \left(\hat{Y}_{i1},\ldots,\hat{Y}_{iL}\right)$ is a vector of size $L$. $\hat{Y}_{ij}$ $(j=1,\ldots,L)$ is the sampled value of the number of $A$ molecules that originated in voxel $i$ at time $t$ and went

to voxel $j$ after a time interval $\tau$. Then

$$\mathbf{Y} = \sum_{i=1}^{L} \hat{\mathbf{Y}}_i \tag{3.19}$$

is a sample of the distribution of $A$ molecules after the diffusion process. Set the population of species $A$ to be $\mathbf{Y}$. Repeat this procedure for each diffusive species.

8. Put the product molecules of the sampled reaction (in step 4) into the sampled voxel (in step 5) where they are produced. Set $t \leftarrow t + \tau$.

9. Return to Step 2, or else stop the realization.

### 3.3.4 Computational cost of the algorithm

As shown in the algorithm, the majority of the computational cost arises from

a. Compute the stepsize $\tau$ (step (2)).

This has been discussed in Section 3.3.1.4, where we found that the computational cost is $O\left(ML^2\right)$.

b. Compute the transition matrix $P(\tau)$ (step (3)).

From equation (A.5.5), the transition matrix is given by

$$\mathbf{P}(t)^T = \mathbf{V}\text{diag}\left(e^{\kappa\lambda_0 t}, \ldots, e^{\kappa\lambda_{L-1} t}\right)\mathbf{V}^T\mathbf{P}(0)^T, \tag{3.20}$$

where $\mathbf{V}$ and $\lambda_0, \ldots, \lambda_{L-1}$ are the normalized eigenvector matrix and eigenvalues of the coefficient matrix in Equation (A.5.1). In the simulation, $\mathbf{P}(0)^T = \mathbf{I}$, thus the computation requires a matrix – diagonal matrix multiplication (cost of

$O\left(L^2\right)$) and a matrix – matrix multiplication (cost of $O\left(L^3\right)$). Although we can reduce the cost by using symmetry properties such as $p_{ij} = p_{ji} = p_{L+1-i,L+1-j} = p_{L+1-j,L+1-i}$, the $O\left(L^3\right)$ complexity still holds.

One way to decrease the complexity is to set a cut-off tolerance for the computation. For example, when we compute $p_{1i}$ $(i = 1, \ldots, L)$, we also record the partial sum of the values that we already computed, i.e. $psum_k = p_{11} + p_{12} + \cdots + p_{1k}$ $(k \leq L)$. If the value $psum_k$ passes some threshold $1 - \epsilon$, then we stop computing and set the remaining variables $p_{1,k+1}, \ldots, p_{1,L}$ to zero. The computed values are then normalized by $p_{1i}\,/psum_k$ $(i = 1, \ldots, k)$, so that they sum up to one. Here $\epsilon$ is a tolerance chosen small enough so that it does not make a noticeable change to the distribution. This strategy can protect us from computing the huge number of very small probabilities when the space is large and the stepsize is small. In our current code, which is used in Section 4 for the numerical experiments, this tolerance is set to be 0 as the default value. However, we still terminate the computation of $p_{1i}$ in two cases: (1). $p_{1i} < 0$; (2). $p_{1i} > p_{1,i-1}$. When these cases occur, it is clear that the numerical precision is no longer reliable, hence the remaining values of $p_{1k}$ $(i < k \leq L)$ may be meaningless.

The time spent in (b) increases linearly with respect to the number of diffusive species, because we need to compute the matrix for each of them. It does not explicitly depend on the number of reaction channels or the molecule populations. However, if these result in an increment in the number of reaction events, the

computational cost will increase since we will need to compute the transition matrix more times.

c. Sample a reaction event (steps (4) and (5)).

Step (4) samples the reaction to fire from the total of $M$ reactions. It requires the time dependent propensity values of the reactions. For example, the propensity of reaction $A + B \xrightarrow{c} C$ can be computed from (3.15). Since we have already computed $\mathbf{V}^T\mathbf{X}^A$ and $\mathbf{V}^T\mathbf{X}^B$ in step (2), computing (3.15) requires only a vector – diagnal matrix – vector multiplication, which is $O(L)$. Since we may, in the worst case, need to compute all of the reaction propensities, the complexity of step (4) is $O(ML)$.

Step (5) samples the original positions of the reactant molecules at the beginning of the step and the location where the reaction occurs. This operation can be done with $O\left(L^2\right)$ cost if the algorithm is carefully designed.

Let us use reaction $A + B \xrightarrow{c} C$ as an example. First we need to sample where the reactant $A$ molecule originates. From the propensity function (3.15), the propensity contributed by the $A$ molecules in voxel 1 is given by

$$\begin{aligned} a^A &= c\left(X_1^A, 0, \ldots, 0\right)\mathbf{V}\mathrm{diag}\left(e^{\left(\kappa^A+\kappa^B\right)\lambda_0\tau}, \ldots, e^{\left(\kappa^A+\kappa^B\right)\lambda_{L-1}\tau}\right)\mathbf{V}^T\mathbf{X}^B \\ &= cX_1^A\mathbf{v}_1^T\mathrm{diag}\left(e^{\left(\kappa^A+\kappa^B\right)\lambda_0\tau}, \ldots, e^{\left(\kappa^A+\kappa^B\right)\lambda_{L-1}\tau}\right)\mathbf{V}^T\mathbf{X}^B, \end{aligned}$$

where $\mathbf{v}_1^T$ is the first row of the matrix $\mathbf{V}$. Thus the probability that the $A$ molecule originates in voxel 1 is $a^A / a_l\left(\tau, \mathbf{X}\right)$, where $a_l\left(\tau, \mathbf{X}\right)$ is the time dependent

propensity of the reaction $A + B \xrightarrow{c} C$, which has already been computed in step (4). Since we have already computed $\mathbf{V}^T\mathbf{X}^B$ in step (2), the computation of $a^A$ basically requires a vector – diagonal matrix – vector multiplication, which is $O(L)$. As the procedure samples over all the voxels in the worst case, it has $O\left(L^2\right)$ complexity, to locate the voxel from which the $A$ molecule originates.

The next task is to locate the voxel from which the $B$ molecule originates. Without loss of generality, let us assume that the $A$ molecule originates in voxel 1. Then among $a^A$, the propensity value contributed by the $B$ molecules from voxel 1 is given by

$$\begin{aligned} a^{AB} &= cX_1^A \mathbf{v}_1^T \mathrm{diag}\left(e^{\left(\kappa^A+\kappa^B\right)\lambda_0\tau}, \ldots, e^{\left(\kappa^A+\kappa^B\right)\lambda_{L-1}\tau}\right) \mathbf{V}^T \begin{pmatrix} X_1^B & 0 & \ldots & 0 \end{pmatrix}^T \\ &= cX_1^A X_1^B \mathbf{v}_1^T \mathrm{diag}\left(e^{\left(\kappa^A+\kappa^B\right)\lambda_0\tau}, \ldots, e^{\left(\kappa^A+\kappa^B\right)\lambda_{L-1}\tau}\right) \mathbf{v}_1. \end{aligned}$$

The probability that the $B$ molecule originates in voxel 1 is $a^{AB} / a^A$. The computation of $a^{AB}$ requires a vector – diagonal matrix – vector multiplication, which is $O(L)$. As the algorithm loops over all the voxels in the worst case, the complexity of sampling the $B$ molecule's position is $O\left(L^2\right)$.

The last task in this step is to sample where the reaction event occurs. Without loss of generality, suppose that both the $A$ molecule and the $B$ molecule originate in voxel 1. Then the probability that the reaction event occurs in voxel $k$ is

$$\frac{cX_1^A X_1^B p_{1k}^A(\tau)\, p_{1k}^B(\tau)}{a^{AB}}.$$

Since we must loop over all the voxels in the worst case, the complexity of this task is $O(L)$.

Putting everything together, step (5) can be implemented with complexity $O\left(L^2\right)$.

d. Sample the diffusion process (step (7)).

As shown in step (7), to redistribute $A$ molecules originating in voxel $i$, we must sample a multinomial random variable, which requires the computation of $L-1$ binomial random variables. Thus, to sample the diffusion process for $A$ molecules originating in every voxel, we must generate $O\left(L^2\right)$ binomial random variables. Once the $L$ multinomial random variables have been generated, we must sum them up as shown in (3.19), which is an $O\left(L^2\right)$ operation. As we need to repeat the procedure for every diffusive species, the complexity of step (7) is $O\left(NL^2\right)$, where $N$ is the number of species. This cost can be reduced if the binomial random variables are sampled in a proper order. For example, to redistribute the $A$ molecules in voxel $i$, we can first sample the number of molecules that will stay in voxel $i$, then the number of molecules that will move to voxel $i-1$, $i+1$, $i-2$, $i+2$,..., until all of the molecules have been redistributed. Thus if the molecules are all distributed in a few voxels near voxel $i$, which is usually the case when the time stepsize is small, the computational complexity of the redistribution can be substantially reduced.

### 3.3.5 Discussion

**The solution of Equation (3.18)**

Newton iteration could be employed to solve Equation (3.18). However, the iteration may fail to converge occasionally due to a bad initial guess. Changing the initial guess is one way to deal with this problem, but it still does not guarantee that we can find a good initial guess in the following trials. Actually, Equation (3.18) has some interesting properties that can help us to find its root. $f(t) = \ln r + \int_0^t a_0(s, \mathbf{X})\, ds$ is a continuous increasing function of $t$, and $f(0) = \ln r < 0$ since r is a uniform random number in $(0, 1)$. Our purpose is to find the root in $(0, T]$, where $T$ is the simulation end time. If $f(T) < 0$, as $f(t)$ is an increasing function, it implies that the root, which is the time to the next reaction event, is not in $(0, T]$. In this case, we can just sample a diffusion process up to time $T$ and finish the simulation. If $f(T) > 0$, then the root is between 0 and $T$. In this case, we first try Newton iteration. If that fails, we use bisection to find the root with a given tolerance $\epsilon$. We first evaluate $f(T/2)$. If $f(T/2) > \epsilon$, we search for the root in $(0, T/2)$. If $f(T/2) < -\epsilon$, we search $(T/2, T]$. If $-\epsilon \leq f(T/2) \leq \epsilon$, we stop the iteration and set $T/2$ to be the root. Since $f(t)$ is continuously increasing, bisection search guarantees that we can find the root.

**Boundary conditions**

In the algorithm as described in this paper, we use reflecting boundary conditions. However, it also works with other boundary conditions as long as one has the closed

form transition probabilities for the corresponding diffusion process. For example, it can be applied with periodic boundary conditions (See Appendix A.5 for the solution of discrete diffusion process with periodic boundary conditions).

**Extension to higher dimension space**

It is straightforward to extend the method to work with a 2D rectangular domain or a 3D cubic domain. For example, on a 2D rectangular domain, the diffusion process in the 'x' direction is independent of the diffusion process in the 'y' direction. Thus the probability for a molecule to jump from voxel $(i_0,\ j_0)$ to voxel $(i_1,\ j_1)$ is the probability that it jumps from column $i_0$ to $i_1$ in the 'x' direction times the probability that it jumps from row $j_0$ to $j_1$ in the 'y' direction.

## 3.4 Numerical simulation

In this section we present some simulation results generated by our new TDPD algorithm and compare with ISSA and NSM simulation results. Computation times of the three methods were obtained on processor Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz with OS windows 7. Here the ISSA method has been implemented with the dependency graph, thus it updates the propensity functions only when necessary. For NSM the dependency graph for reactions has been implemented, as well as the strategy to reuse the random number for voxels that receive molecules from the neighbours

| Method | Realizations | Resolution | Average time per realization |
|---|---|---|---|
| ISSA | 100000 | 2 voxels | 0.02756s |
| TDPD | 100000 | 2 voxels | 0.00121s |
| NSM | 100000 | 2 voxels | 0.02979s |
| ISSA | 10000 | 50 voxels | 51.3864s |
| TDPD | 10000 | 50 voxels | 0.13936s |
| NSM | 10000 | 50 voxels | 41.388s |

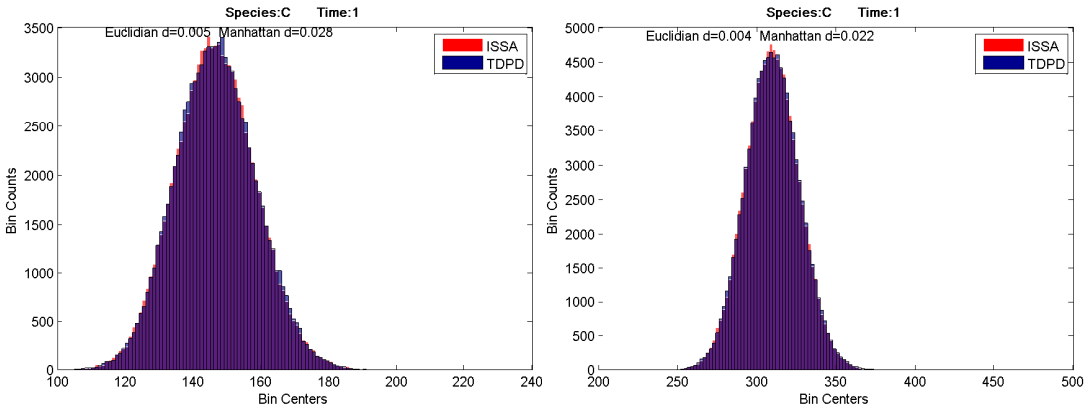
Table 3.1: CPU times for the one second simulation of Example 1. The first three entries use a resolution of two subvolumes. The last three entries use a resolution of 50 subvolumes.

[32]. In addition, the software package MesoRD is used in Example 2 for comparison purposes.

### 3.4.1 Example 1

This example is from Section 3.3. It consists of two voxels and one reaction $A + B \xrightarrow{c} C$. The initial values used for the simulation were: 10000 $A$ molecules in the first voxel; 10000 $B$ molecules in the second voxel; no $C$ molecules. The reaction rate constant is $10^{-5}$. The diffusion rates for species $A$, $B$ and $C$ are 10, 1, 0.1 respectively. The simulation time is 1 second. The first three entries in Table 3.1 show the CPU time used for the simulation, where it is apparent that the TDPD method achieves an order of magnitude speedup over ISSA and NSM.

Histograms of species C in the two voxels at time $t = 1s$ are shown in Fig. 3.1, and reveal that the new TDPD algorithm is quite accurate. At the top of each figure we provide two values to measure the difference of the histograms. The definitions of the two measures are as follows. Let $\mathbf{X} = (x_1, \ldots, x_n)$ be a vector that corresponds




(a) C molecules in the first voxel (b) C molecules in the second voxel

Figure 3.1: Histograms of species C. Comparison of results given by ISSA and TDPD. Red is ISSA, blue is TDPD, and purple is the overlap of the two histograms.

to a histogram where $x_i$ is the count in bin $i$, and $\mathbf{x} = \mathbf{X} / \sum_{i=1}^{n} X_i$ is the normalized $\mathbf{X}$. Then for two histograms, we have two normalized vectors $\mathbf{x}$ and $\mathbf{y}$. The Euclidean distance in the histogram figures is defined as the 2-norm of $\mathbf{x}-\mathbf{y}$, i.e. $\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$. The Manhattan distance is defined as the 1-norm of $\mathbf{x} - \mathbf{y}$, i.e. $\sum_{i=1}^{n} |x_i - y_i|$.

For the next test, we increased the resolution of the one dimensional model from 2 voxels to 50 voxels. The diffusion and reaction rates also changed due to the change of the subvolume size (i.e. the diffusion rates increased $25^2$ times, and the reaction rate increased 25 times). Initially the $A$ and $B$ molecules were located in the two boundary voxels of the one dimensional geometry respectively. The last three entries in Table 3.1 show the CPU times used for the simulations. Figure 3.2 shows the average population of species $C$ in each voxel. The TDPD and NSM methods generate nearly identical results.

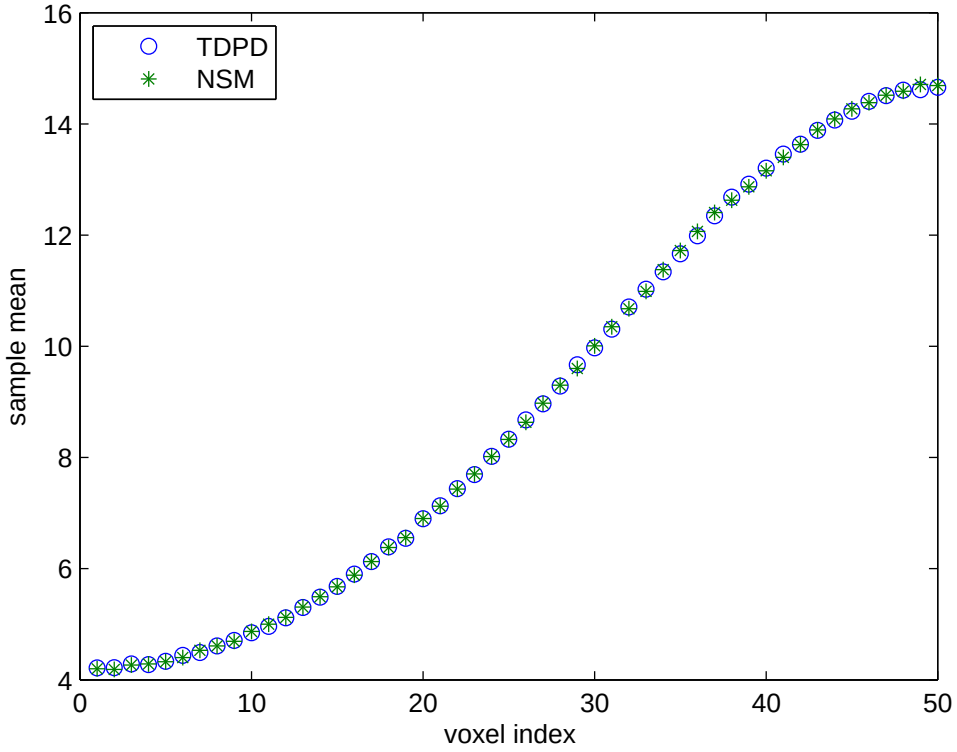
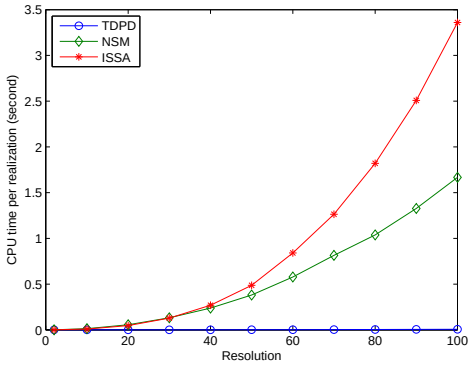



Figure 3.2: Average population of species C in each voxel at $t = 1$. The resolution is 50 voxels. 10000 realizations are simulated for each method.

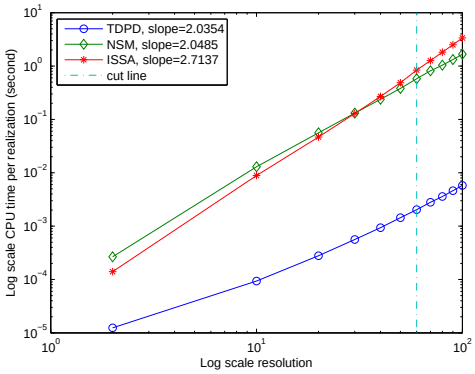
In addition to the accuracy, we are interested in the computation time of the algorithm. In the next subsection we will demonstrate how the simulation time scales with the resolution, the species population and the number of reaction channels for this example.

#### 3.4.1.1 Scaling of simulation time with respect to resolution

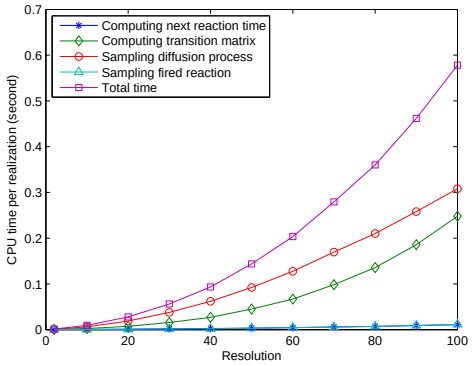
In the previous experiments, we have run the simulation with resolution of 2 and of 50 voxels. In order to show how the simulation time scales with respect to the



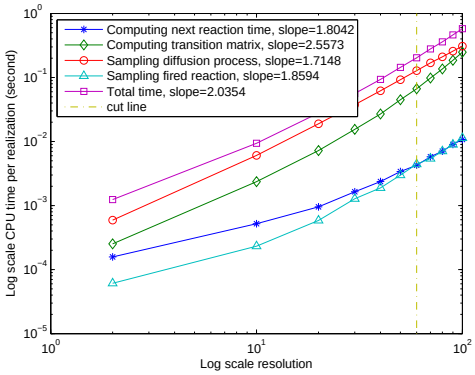

(a) CPU time used for one realization of Example 1 under different resolutions.




(b) Log-log plot of Figure 3.3a. The slopes are computed from the points on the right hand side of the cut line.



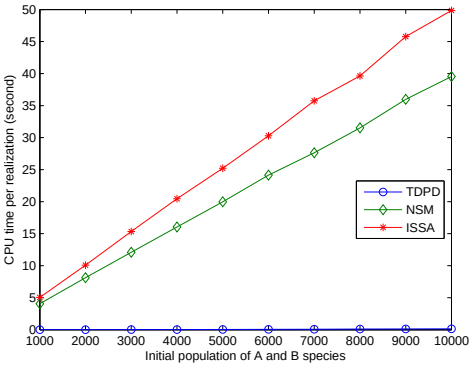

(c) CPU time used by the four operations in the TDPD algorithm in one realization under different resolutions.



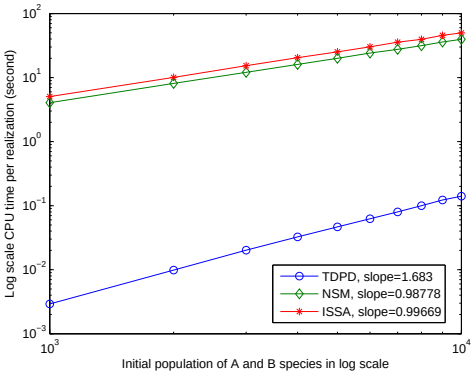

(d) Log-log plot of Figure 3.3c. The slopes are computed from the points on the right hand side of the cut line.

Figure 3.3: Scaling of computation time with respect to resolution.

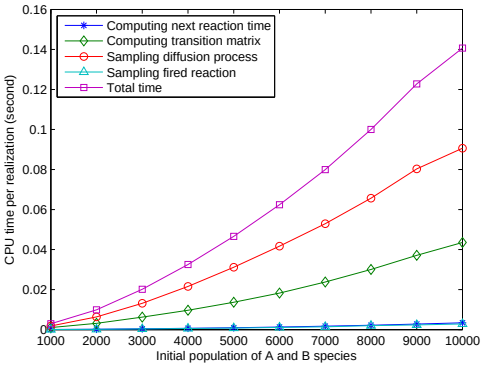
resolution, we also ran the simulation with resolutions of 10, 20, 30, ..., 100 voxels. For each resolution, there are initially 10000 $A$ and $B$ molecules in the two boundary voxels respectively, and 1000 realizations are run with TDPD and with ISSA and NSM for comparison. Figure 3.3 shows the average CPU time used for one realization. Figure 3.3a shows that TDPD enjoys an orders of magnitude performance increase over ISSA and NSM. Figure 3.3b is the log scale plot of Figure 3.3a. It shows that the TDPD and NSM have similar slopes, which are better than the ISSA's slope. As we have discussed in Section 3.3.4, there are four operations in the TDPD algorithm that occupy the majority of computation time. Figure 3.3c plots the time used by the four operations in each realization under different resolutions. It reveals that sampling the diffusion process (step (7) in the algorithm) is the most expensive operation. The next expensive operation is computing the transition matrix (step (3) in the algorithm). Computing the next reaction time (step (2)) and sampling a reaction event (step (4) (5)) are much cheaper than the previous two operations (they are overlapped in Figure 3.3c). Figure 3.3d shows the log scale plot of Figure 3.3c. Note that even though computing the transition matrix is cheaper than sampling the diffusion process in Figure 3.3c, it has a larger slope in the log-log plot; thus it may be the most expensive operation when the resolution is very high.



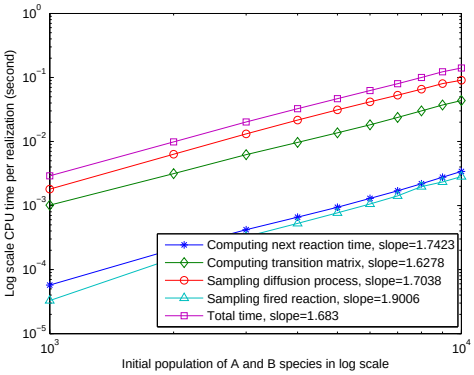

(a) CPU time used for one realization of Example 1 with different initial populations.




(b) Log-log plot of Figure 3.4a.




(c) CPU time used by the four operations in the TDPD algorithm in one realization with different initial populations.




(d) Log-log plot of Figure 3.4c.

Figure 3.4: Scaling of computation time with respect to the initial population. Values are averaged over 1000 realizations.

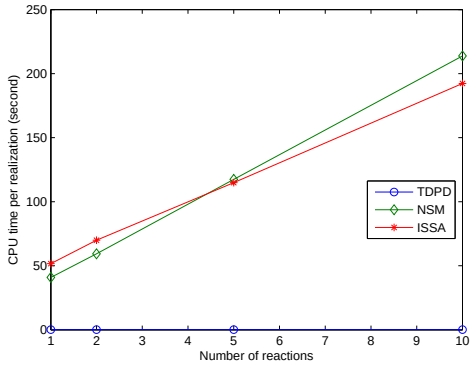
#### 3.4.1.2 Scaling of simulation time with respect to species' population

In the previous experiments, we initially have 10000 $A$ molecules and 10000 $B$ molecules in the two boundary voxels respectively. In this subsection, we run a set of simulations with initially 1000, 2000, 3000, ..., 10000 $A$ and $B$ molecules in the two boundary voxels respectively. The resolution is set to be 50 voxels. Figure 3.4 shows the computation times. Figure 3.4a plots the CPU time used by ISSA, NSM, and TDPD, for one realization with different initial populations. TDPD performs the best of the three. Figure 3.4b is the log-log plot of Figure 3.4a. It shows that ISSA and NSM have a slope near one while TDPD has a slope greater than one. This result can be explained as follows: In a system where the number of diffusion events overwhelms the number of reaction events, when the population of $A$ and $B$ molecules increases $k$ times, the number of diffusion events in the system will also increase roughly $k$ times. Thus ISSA and NSM must take roughly $k$ times more steps to run the simulation, which explains why Figure 3.4a and 3.4b shows a linear relationship between ISSA, NSM and the initial population. In contrast, the computation time of TDPD is immune from the impact of diffusion. It filters the massive linear increment of diffusion events. However, it must still deal with the increment from reaction events. As the populations of both $A$ and $B$ increase by $k$ times, the time dependent reaction propensity increases $k^2$ times at the beginning the simulation, which explains why the computation time of TDPD has a slope larger than one in Figure 3.4b. Figures 3.4c and 3.4d show the time used by the

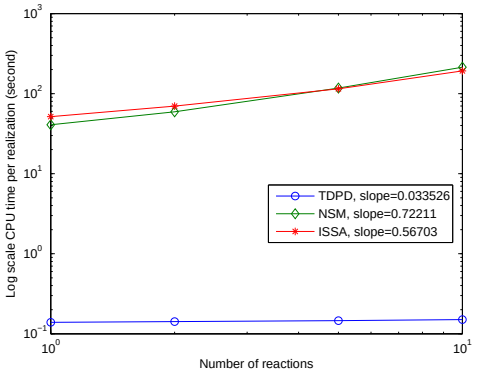
four main operations in TDPD. Figure 3.4d shows that the four operations have similar slopes.

#### 3.4.1.3 Scaling of simulation time with respect to the number of reaction channels

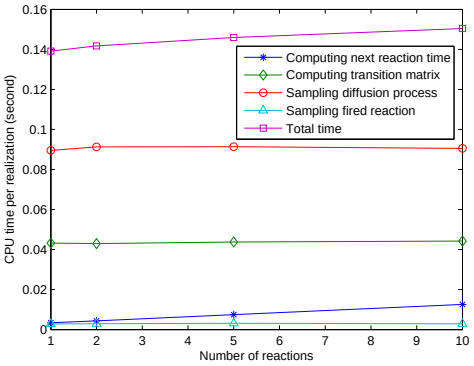
In this subsection we ran a set of simulations with the reaction channel copied $k$ ($k = 1, 2, 5, 10$) times. For example, when $k = 2$, the system has two reactions, both of which have the form $A + B \xrightarrow{c_k} C$. We set the reaction rate $c_k = c/k$ for all the reaction channels, where $c = 10^{-5}$ is the original reaction rate. All of the simulations should have a similar number of reaction events; thus the number of reaction channels will be responsible for the change of computation times. For all the simulations we set the resolution to be 50 voxels, with initially 10000 $A$ molecules at one end, and 10000 $B$ molecules at the other end. The computation times are shown in Figure 3.5. Figure 3.5a shows that the computation time for ISSA and NSM increases significantly with respect to the number of reaction channels. The log-log plot of Figure 3.5b shows that the slope of the computation time of TDPD is much smaller than one, which means that the increase in the number of reaction channels has little influence on the simulation cost of TDPD. Further decomposition of the computation time in TDPD are shown in Figures 3.5c and 3.5d. As the number of species is the same for all the simulations, computing the transition matrices and sampling the diffusion processes take a similar amount of time for each simulation. The time spent in sampling the reaction events (steps 4 and



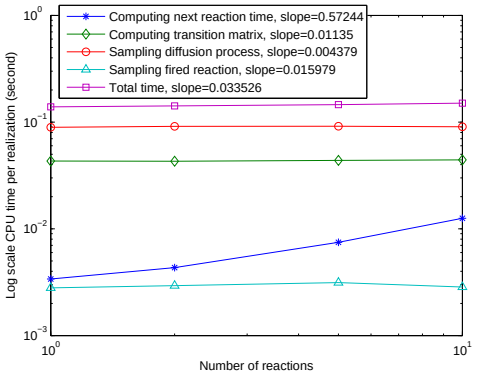

(a) CPU time used for one realization of Example 1 with different numbers of reaction channels.




(b) Log-log plot of Figure 3.5a.




(c) CPU time used by the four operations in the TDPD algorithm in one realization with different numbers of reaction channels.




(d) Log-log plot of Figure 3.5c.

Figure 3.5: Scaling of computation time with respect to the number of reaction channels. Values are averaged over 1000 realizations.

5 in the algorithm) is influenced by the number of reaction channels. In step 4 the index of the reaction channel is sampled, and in step 5 the locations where the reactant molecules originate and where the reaction event occurs are sampled. As analyzed in Section 3.3.4, the leading term of the complexity comes from step 5, which does not depend on the number of reaction channels. Thus the corresponding curve in Figure 3.5c looks almost flat. The time spent on computing the next reaction time, however, has a strong relationship with the number of reaction channels. This is because when we solve (3.18), we need to compute the time dependent propensity for every reaction channel; thus the more channels we have, the more values we need to compute. Figure 3.5c shows that this part is responsible for almost all of the increase in computation time in the TDPD simulation.

### 3.4.2 Example 2

Example 2 is a two dimensional problem with $3 \times 3$ voxels. The chemistry consists of the following first and second order reactions:

$$S_0 + S_0 \xrightarrow{c_1} S_1, \quad S_1 \xrightarrow{c_2} S_0 + S_0, \quad S_0 + S_1 \xrightarrow{c_3} S_2, \quad S_2 + S_3 \xrightarrow{c_4} S_3 + S_4$$

To make the example spatially inhomogeneous, we begin with one $S_3$ molecule, which is fixed in the bottom right corner. Thus an $S_2$ molecule can be converted to $S_4$ only when it travels to the bottom right voxel and reacts with the $S_3$ molecule.

| Method | Realizations | Average time per realization |
|---|---|---|
| ISSA | 100000 | 8.65476s |
| NSM | 100000 | 8.13928s |
| TDPD | 100000 | 0.09501s |

Table 3.2: Computation time for the ten second simulation of Example 2.

Initially we have 10000 $S_0$ molecules in the top left corner. The rate parameters used in the simulation are given by

$$c_1 = 10^{-4}, \quad c_2 = 0.1, \quad c_3 = 0.01, \quad c_4 = 0.1,$$

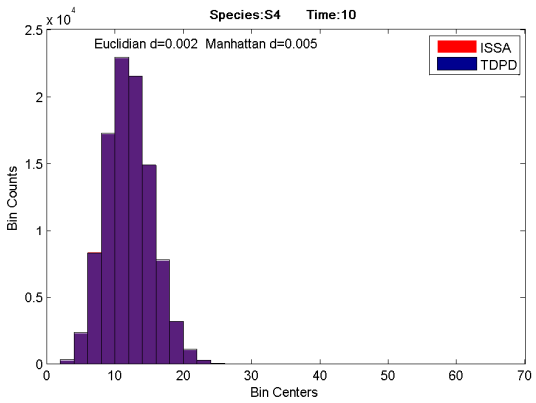
and the diffusion rates for the species are given by

$$S_0 : 100, \quad S_1 : 10, \quad S_2 : 5, \quad S_3 : 0, \quad S_4 : 1.$$

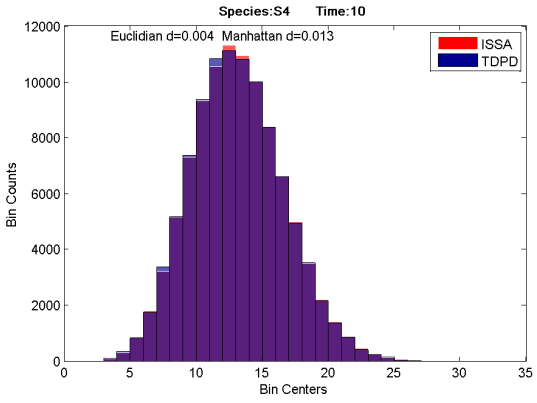
The time used for a ten second simulation is shown in Table 3.2. The new algorithm has a significant speedup over ISSA and NSM.

To demonstrate the accuracy of our algorithm, we plotted the histograms of the product $S_4$ in voxels (1,1), (1,2), (1,3), (2,2), (2,3), (3,3) (Here voxel $(i, j)$ means the voxel at row $i$ and column $j$), together with the distribution given by ISSA, in Figure 3.6. It is evident that our algorithm can produce very accurate results.

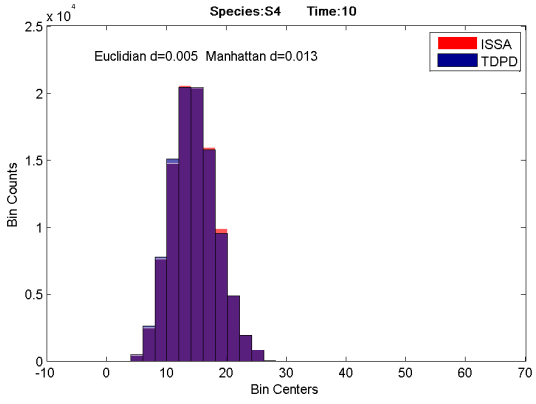
For this model we have also increased the resolution to compare the performance of different methods. Figure 3.7 shows the CPU times used by different methods for one realization of Example 2. It is evident that TDPD enjoys substantially better performance than the other methods. Figure 3.7c is the log scale plot of the CPU times.



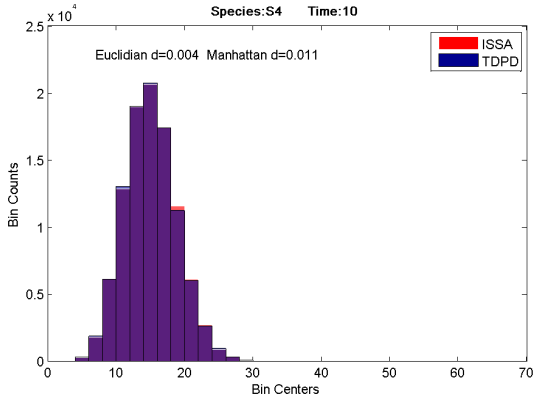

(a) S4 molecules in the voxel (1,1)



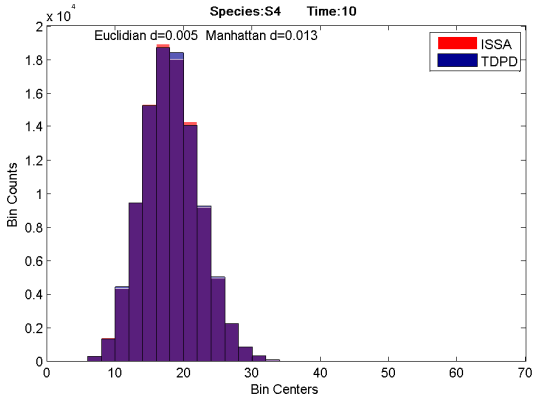

(b) S4 molecules in the voxel (1,2)



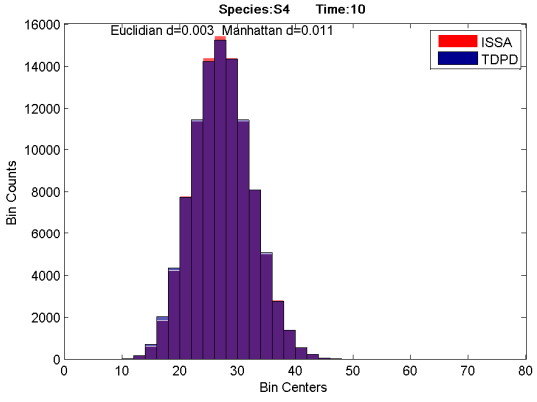

(c) S4 molecules in the voxel (1,3)




(d) S4 molecules in the voxel (2,2)

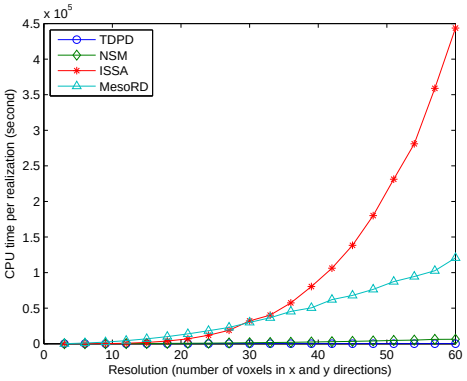



(e) S4 molecules in the voxel (2,3)

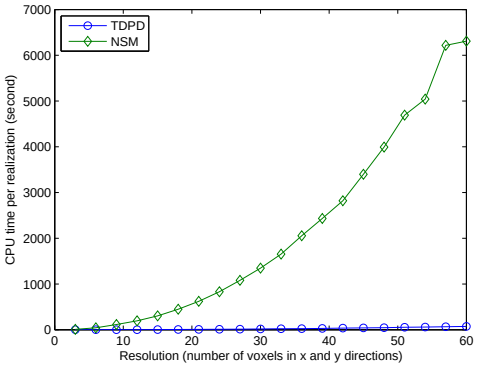



(f) S4 molecules in the voxel (3,3)

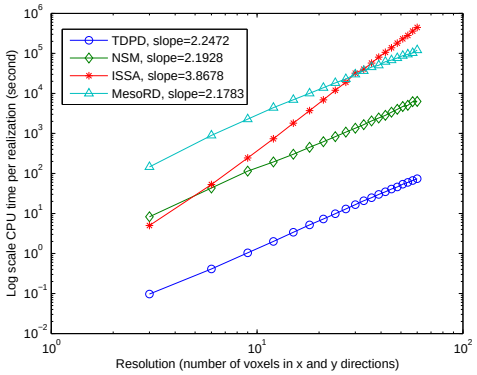
Figure 3.6: Histograms of species $S_4$. Comparison of result given by ISSA and TDPD. Red is ISSA, blue is TDPD, and purple is the overlap of the two histograms.




(a) CPU times used by TDPD, NSM, ISSA and MesoRD for one realization of Example 2 under different resolutions.




(b) Zoom in on the TDPD and NSM curves from Figure 3.7a.




(c) Log-log plot of Figure 3.7a

Figure 3.7: Scaling of computation time with respect to resolution.

It shows that the computation time of the TDPD method has a similar slope as the NSM and MesoRD, which is smaller than the ISSA's slope.

### 3.4.3 Example 3: Demonstration of the error behavior of the TDPD method

In Section 3.3.1.3 we noted that the error of the simulation might be large when $E\left(p_r\left(\tau\right)\right)$ is large, where $E\left(p_r\left(\tau\right)\right)$ is bounded by (3.11). It is evident that when the total propensity of the system $a_0(t)$ is much larger than the propensity contributed by a single molecule $a(t)$, the right hand side of (3.11) will be small, thus the simulation will have good accuracy. In this section we will use an example to demonstrate this point.

Suppose that we have a one dimensional system with absorbing boundary conditions, with a population of $A$ molecules that are initially in the central voxel. There are 50 voxels on both sides of the central voxel. The diffusion coefficient is set to be 300 for the simulation. The simulation time is one second.

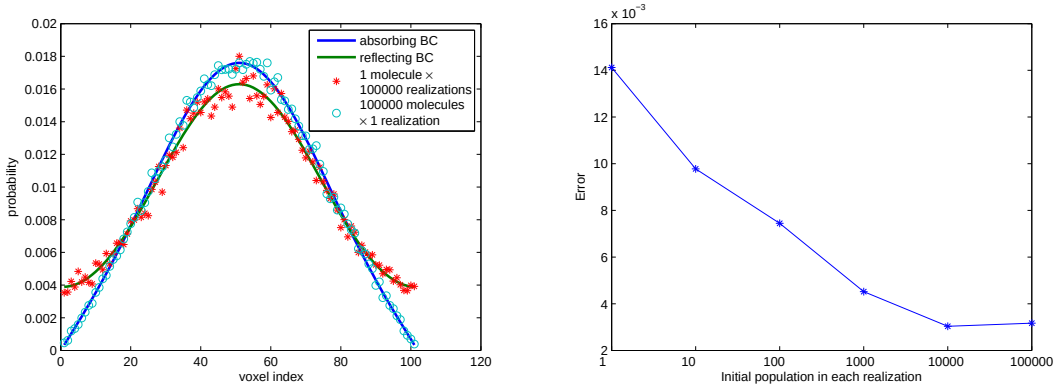
In order to perform the simulation with our algorithm, we modified the system slightly by replacing the escaping diffusion events in the two boundary voxels by absorbing reaction events $A + B \xrightarrow{c} B$, where we put one non-diffusive $B$ molecule in each boundary voxel and the reaction rate is also set to be $c = 300$. It is obvious that the modified system is virtually equivalent to the previous diffusion system with absorbing boundary condition (since species $A$ is governed by the same reaction-diffusion

master equation in the two systems), and the "$B$ molecules" are actually the holes in the boundary that allow molecules to escape. Now we have a diffusion system with reflecting boundary conditions, plus an absorbing reaction in the two boundary voxels.

We chose this example in part because its analytical solution is available (See Appendix A.5). Thus, it is convenient for us to compare the numerical solution with its true solution for error analysis purposes. In this section, we will discuss how the number of molecules, geometry resolution, and number of reaction channels affect the accuracy.

#### 3.4.3.1 Accuracy with respect to the number of molecules

Equation (3.11) indicates that the simulation may incur a large error when the total propensity $a_0(t)$ is not large compared with the propensity contributed by a single molecule. We can maximize this error by pushing it to an extreme in which the system involves only one molecule initially, thus $a_0(t) = a(t)$. In this case, the algorithm will directly sample the time of the absorbing reaction. If it is larger than the terminating time, the molecule survives and its location will be sampled according to a diffusion process with reflecting boundary conditions, as stated in the algorithm. Thus after 100,000 realizations we obtain a distribution of results (shown in Figure 3.8a) which suggests that the location of the surviving molecules are that of a diffusion process with reflecting boundary conditions. However, this is obviously not correct, since we




(a) (b)

Figure 3.8: (a) Simulation results for Example 3. The blue line is the analytical solution of a diffusion process with absorbing boundary conditions. The green line is the analytical solution of a diffusion process with reflecting boundary conditions. The red stars are from 100,000 realizations with one molecule initially. The circles are from one realization with 100,000 molecules initially. (b) Errors for simulations with different initial populations.

know that the solution should be that of a diffusion process with absorbing boundary conditions.

Next we created another simulation for comparison. We initially put 100,000 molecules in the central voxel and performed only one realization. This time the total propensity of the system is much larger than the propensity contributed by a single molecule. The analysis in Section 3.3.1.3 predicts that the simulation should give us a much better result. Figure 3.8a verifies that this simulation generates a distribution which is quite close to the diffusion process with absorbing boundary condition.

In order to quantitatively show how the error changes with respect to the number of molecules in the simulation, the following simulations were also performed: 10,000 realizations with initial population 10 molecules; 1000 realizations with initial popula-

tion 100 molecules; 100 realizations with initial population 1000 molecules; and 10 realizations with initial population 10,000 molecules. Each experiment has the same number of molecule samples and generates a probability distribution $\mathbf{p}_{simulate} = (p_1, \ldots, p_L)$, where $p_i$ is the probability that a survived molecule is observed in voxel $i$. Comparing this result with the analytical solution $\mathbf{p}_{analytical}$ computed from (A.5.12), we can obtain the error of the simulation as

$$\text{Error} = \|\mathbf{p}_{simulate} - \mathbf{p}_{analytical}\|_2 . \tag{3.21}$$

Figure 3.8b shows the errors in each simulation. As expected, the error decreases as the initial population increases.

This result can also be explained from another point of view. In each step of the simulation, the algorithm uses the diffusion process with reflecting boundary conditions to approximate the true distribution, which in this example is the diffusion process with absorbing boundary conditions. The true distribution and our approximation start from the same initial condition and diverge as time goes on. Thus the error increases as the stepsize increases. This is quite like using the explicit Euler method for solving ODEs, which uses a straight line to approximate the true solution curve in each step. In the simulation with one molecule in the system, a realization involves at most one reaction event, thus it needs only one step to finish the simulation. As a result, the stepsize is very large and the error will be significant. On the other hand, in the simulation with 100,000 molecules, 7375 reaction events occur. Thus the average stepsize is roughly $1.36 \times 10^{-4}$s, which significantly reduces the error of the simulation.

#### 3.4.3.2 Accuracy with respect to the resolution

In the previous simulations, we have 50 voxels on each side of the central voxel, i.e. 101 voxels in total. In order to explore how the accuracy changes with respect to the resolution, we ran another set of simulations with resolution of 21 voxels, 41 voxels, ..., 101 voxels. For each simulation we put 100,000 $A$ molecules in the central voxel initially. The absorbing reactions that occur in the two boundary voxels have a reaction rate that has been set equal to the diffusion rate, which is updated for each simulation due to the change of resolution. Ten realizations are run for each parameter (so there are 1,000,000 molecule samples in total for each simulation). The error of each simulation is computed as in (3.21). Here the analytical solution is computed with $L = 21, 41, \ldots, 101$ respectively. Table 3.3 shows the error under different resolutions. The table suggests that the error does not change much when the resolution changes. As far as (3.11) is concerned, it means that the ratio between the propensity contributed by a single molecule and the total propensity of the system is similar for each simulation. In this simple system, it implies that in each simulation the total number of molecules that remain in the system is at the same level. The last entry in Table 3.3 shows the number of survived molecules in each simulation. As we expected, the number of molecules that survived the one second experiment is similar for each simulation with different resolutions. This result agrees with our intuition: the number of molecules that escape the one dimensional channel is a property of the system, which should not depend on the resolution used by a simulation.

| Resolution | 21 | 41 | 61 | 81 | 101 |
|---|---|---|---|---|---|
| Error $(\times 10^{-4})$ | 9.4304 | 8.3135 | 9.1124 | 9.8801 | 10.724 |
| Survived molecules | 93654.8 | 92969.4 | 92696.4 | 92608.2 | 92454.2 |

Table 3.3: Simulation error under different resolutions. Ten realizations are used for each parameter.

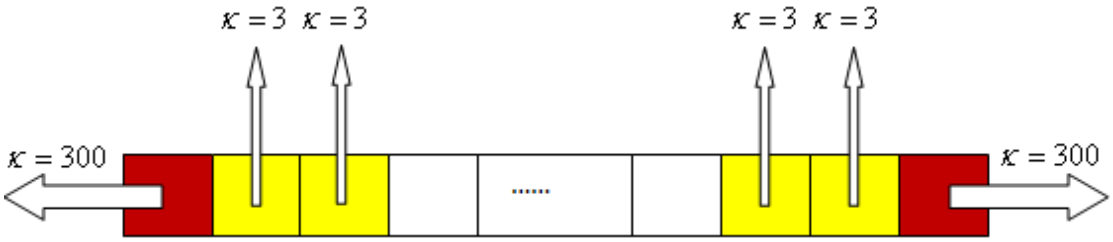



Figure 3.9: Absorbing reaction channels are added in the yellow voxels, whose reaction rates are set to be 3.

#### 3.4.3.3 Accuracy with respect to the number of reaction channels

In the previous experiments, molecules can only escape the system from the boundary voxels. In order to show how the error changes with respect to the number of reaction channels, we will run simulations with more and more voxels that have absorbing reaction channels in them. I.e. we drill holes on more and more voxels in the channel. Figure 3.9 shows how the experiments are designed. For all the simulations we use 101 voxels as the resolution. The initial population in the central voxel is 100,000 molecules. The diffusion rate is 300. The yellow voxels in the figure have absorbing reaction channles in them, whose reaction rates are set to be 3. We will set more and more voxels to be yellow from the two ends of the channel, thus the simulations will have 10, 20, 30, 40, 50 voxels on each end having absoring reactions (including the red voxel at the boundary). Ten realizations are used for each parameter.

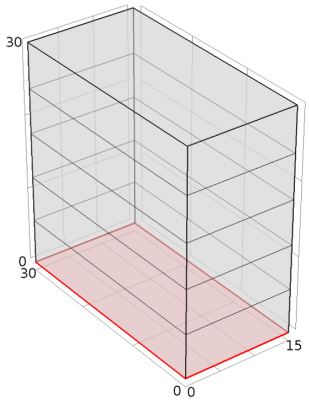
| Number of absorbing reaction channels | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|
| Error $(\times 10^{-3})$ | 1.0277 | 1.1830 | 1.3467 | 1.9767 | 4.1608 |
| Survived molecules | 90657.6 | 82230.9 | 63224.1 | 30961.2 | 5152.4 |

Table 3.4: Simulation error with different number of reaction channels. Ten realizations are used for each parameter.

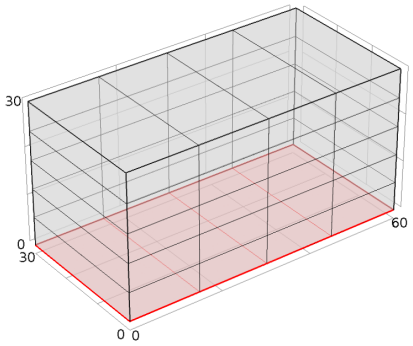
Since for this example the analytical solution is no longer easy to compute, we use the simulation result from exact methods (here we use NSM) instead. Table 3.4 shows the errors of the simulations. It reveals that the error has an increasing trend as the number of absorbing channels increases. This trend can also be explained by equation (3.11). The more absorbing channels the system has, the fewer molecules remain in the system. Thus the ratio between the propensity contributed by a single molecule and the total propensity of the system will increase, which implies that the error of the simulation will increase as well. The last entry in Table 3.4 supports our reasoning: the number of molecules survived the one second simulation decreases significantly as the number of absorbing reaction channel increases.

### 3.4.4 Coagulation model

The final example is a widely used model of blood coagulation [30] with 43 reactions and 33 species. When a blood vessel is wounded, it exposes Tissue Factor (TF) on the wounded vessel surface. TF initializes the extrinsic pathway of the coagulation cascade, which generates thrombin in the vessel. Thrombin then activates platelets, which form clots to prevent the loss of blood (the latter process is not modeled here).




(a) A control volume of $15 \times 30 \times 30\,(\mu m)^3$.




(b) A control volume of $60 \times 30 \times 30\,(\mu m)^3$, which is four times the volume in (3.10a).

Figure 3.10: The geometry of the control volumes for our simulation. The red surface at the bottom represents the wounded blood vessel surface which contains TF.

The original ODE model has for its state variables the concentration of each species as opposed to population, which is tracked in discrete stochastic simulation. We converted the concentration to population by selecting a control volume as shown in Figure 3.10. The bottom surface (the red surface in Figure 3.10) represents the wounded blood vessel. We begin with a control volume of $30\mu$m $\times$ $30\mu$m $\times$ $15\mu$m (Figure 3.10a), where the $30\mu$m $\times$ $30\mu$m area is of the same level as the cross section of a capillary. The diffusion rates are set to be $50\mu\text{m}^2/\text{s}$ for every species. Since the workload of the simulation is very heavy, it is important for us to reduce the complexity of the model. As the spatial inhomogeneities arise mainly from the species and reactions that belong to the wounded surface, we assume that the system is homogeneous in the 'x' and 'y' directions but inhomogeneous in the 'z' direction. Thus we discretize space in the z

| Method | Control volume length × width$^2$ ($\mu$m$^3$) | Realizations | Average time per realization |
|---|---|---|---|
| TDPD | $15 \times 30^2$ | 60 | 3745s |
| ISSA | $15 \times 30^2$ | 1 | 56403s |
| NSM | $15 \times 30^2$ | 1 | 51519s |
| TDPD | $60 \times 30^2$ | 30 | 84420s |

Table 3.5: The time used for the 700 second simulation of the coagulation model.

direction, yielding a 1D model. In the simulation, we divide the space into five voxels along the z axis. Since TF appears only on the wounded vessel surface, we assume that TF and any compound involving TF exists only in the bottom voxel, and does not diffuse upward. In this example the ISSA simulation is extremely slow (Table 3.5 shows the ISSA speed). Thus we will compare the results of our method to a PDE simulation (i.e. we compare the dynamics of mean thrombin concentration from the stochastic simulation to the PDE result). Both stochastic and PDE models use the same height of 30$\mu$m for the control volume. However, intuition tells us that the larger the control volume, the less the stochastic effect will be. Thus we show the results for another stochastic simulation which increases the length of the control volume (Figure 3.10b). We expect that the stochastic simulation result should approach the PDE result, as the control volume gets larger.

The times required for the 700 second simulation are shown in Table 3.5. Due to the huge number of molecules, the simulation of diffusion events makes ISSA slow for this model. However, by using the time dependent propensity function in the simulation

to avoid sampling of individual diffusion events, we can obtain simulation results at a greatly reduced computational cost.

Figure 3.11 shows the mean values (over space and over all realizations) of the thrombin concentration given by the stochastic simulations and the concentration given by the PDE solution.

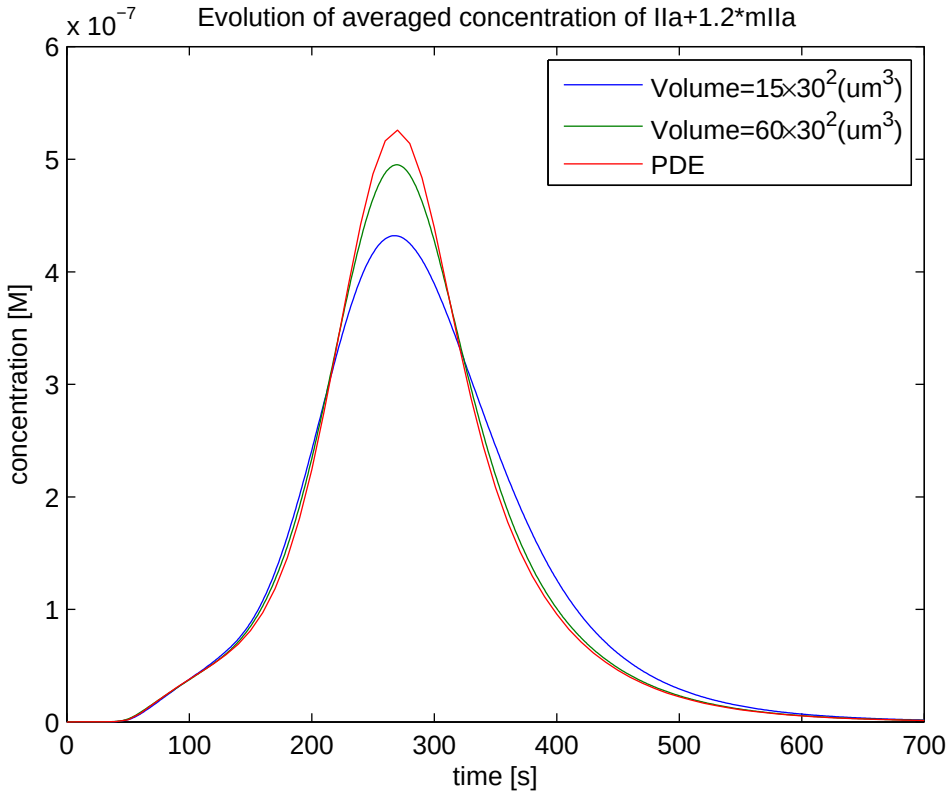
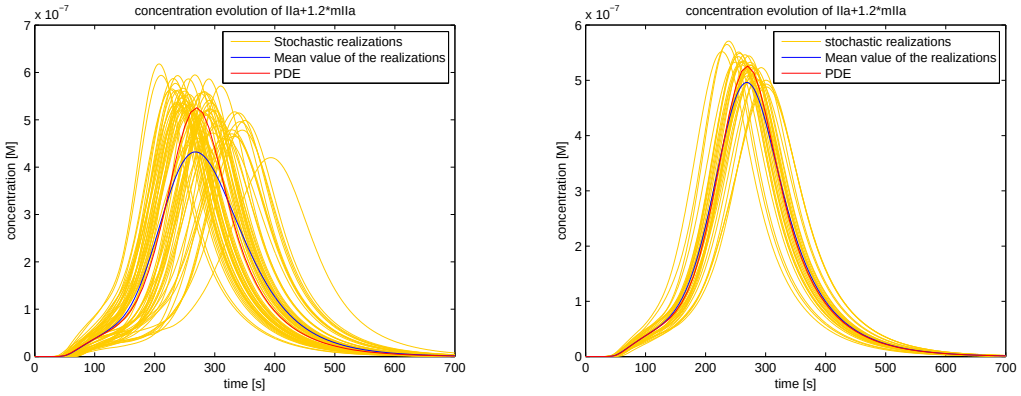



Figure 3.11: Dynamics of the averaged thrombin concentration for different control volumes. Here IIa is activated thrombin, and mIIa is meizothrombin which is an intermediate that is produced during the conversion of prothrombin to thrombin.

The trend of the curves follows our expectations. It is evident from the figure that when the control volume is small, the peak value of the average thrombin response




(a) 60 stochastic realizations with control volume $15 \times 30^2(\mu m^3)$.

(b) 30 stochastic realizations with control volume $60 \times 30^2(\mu m^3)$.

Figure 3.12: Stochastic realizations and their averaged responses

is low. As the control volume increases, the averaged response is approaching that of the PDE solution. An explanation of the result is the self-propagation of thrombin. Thrombin can accelerate its formation by activating other factors which can form catalysts for thrombin generation. This can also be observed from the PDE curve in Figure 3.11. (Initially the curve has a small slope; as the concentration of thrombin increases, the slope of the curve increases dramatically). However, in the stochastic model, the situation is more complex.

Due to stochastic effects, the initialization time of thrombin response differs among realizations. This can be easily observed if we plot all the trajectory curves. Figure 3.12a shows that when the control volume is small ($15 \times 30^2 \mu \mathrm{m}^3$), the variation between different realizations can be significant. This variation leads to the fact that the average of the realization curves has a wider bell shape with a lower peak value (the blue curve in Figure 3.12a) compared with the PDE solution (the red curve in Figure

3.12a). When we increase the control volume ($60 \times 30^2 \mu m^3$), as shown in Figure 3.12b, the variation between realizations becomes smaller. As a result, the bell shape mean response curve becomes narrower and higher (the blue curve in Figure 3.12b), which more closely matches the PDE curve.

## 3.5 Conclusion

Spatial stochastic simulation using the time dependent propensity provides a means to accelerate the simulation of systems whose diffusion events overwhelm reaction events. The key point of the method is that it uses the time between adjacent reaction events as the simulation stepsize; individual diffusion events during the step are not tracked. However the effect of the diffusion process is still accounted for by using the time dependent propensity functions for each reaction. Thus the method yields a speedup by avoiding the sampling of the individual diffusion events, while still maintaining excellent accuracy. The idea of the method can also be easily extended for simulations of 2D rectangular regions and 3D cuboid regions.

However, the method still has some limitations.

1. It accelerates the simulation only when the number of diffusion events is much larger than that of the reaction events. When this condition does not hold, the overhead of computing time dependent propensity functions will slow down the simulation compared to an exact method.

2. In the algorithm, a molecule is allowed to diffuse to any subvolume in one step. However in some cases, it is more likely that a molecule walks in a local region as opposed to traversing the whole space during a stepsize. Thus, keeping track of the molecule in a truncated space may greatly decrease the computational cost. As future work, we have designed an algorithm that implements this idea and a general purpose code is now under development.

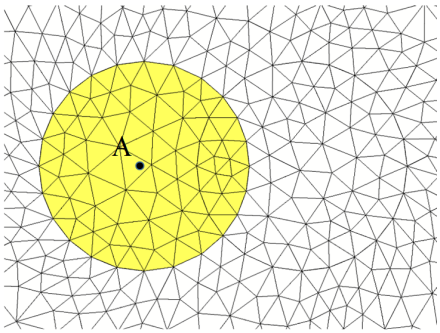
3. For arbitrary geometry or unstructured meshes, the closed form solution of the probabilities that one molecule jumps from one voxel to another voxel may not be easy to obtain. We may need to use approximation functions (e.g. compute the value at some time points and then do interpolation) in these cases. It might be helpful to store these values so that they can be reused in the simulation.

# Chapter 4

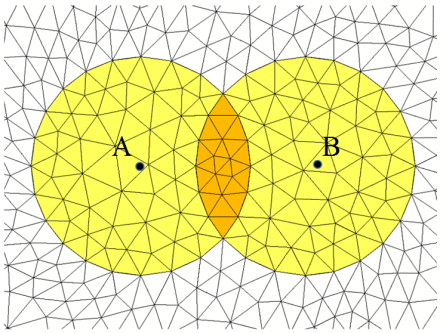
# Time dependent propensity for diffusion (TDPD) method on unstructured mesh

## 4.1 Introduction

In Chapter 3, the TDPD algorithm on a regular mesh in rectangular domains was presented. In this chapter, we extend the algorithm to unstructured mesh.




(a) Region of the space that a molecule can diffuse to in one step (the yellow voxels).




(b) Two molecules can react if both of them diffuse to the same orange voxel.

Figure 4.1: Demonstration of the FSP

## 4.2 TDPD on unstructured mesh

This section demonstrates how to apply the TDPD algorithm on an unstructured mesh. We begin with the difference between an unstructured mesh and a regular mesh.

### 4.2.1 Difference between regular and unstructured mesh

In Chapter 3, a regular mesh was used for the TDPD algorithm. One advantage of the regular mesh in a rectangular domain is that the transition probability from one voxel to another has a closed form solution, which simplifies the computation of the time dependent propensity. However, on an unstructured mesh we lose such convenience. In order to decrease the computational cost, we need to restrict the space into which a molecule can diffuse in one step (as shown in the yellow region in Figure 4.1a). In simulations, this region may not necessarily to be a circle.

Suppose the system has only one reaction channel $A + B \to \phi$, and the current time is 0. As shown in Figure 4.1b, a molecule of species A in voxel $i$ and a molecule of species B in voxel $j$ can diffuse to the nearby yellow voxels in one step with stepsize $\tau$. The overlap of the two regions is shown in orange. If both of the two molecules diffuse to the same voxel in the orange region, they have a chance to fire a reaction. Denote by $a_{ij}(t)$ the propensity function of the reaction at time $t \leq \tau$. Then $a_{ij}(t)dt$ is the probability that the two molecules will react in the infinitesimal time interval $[t, t + dt]$, given that they still survive at time $t$. Under the restriction that the molecules can diffuse only within the yellow region, $a_{ij}(t)dt$ can be represented as

$$\begin{aligned} a_{ij}(t)dt &= \mathbf{P}(\text{the two molecules react in } [t, t + dt]) \\ &= \mathbf{P}(\text{they are in the same voxel with orange color at } t) \times \mathbf{P}(\text{they react in } [t, t + dt]) \\ &= \sum_{k \in \text{orange region}} P_{ik}^A(t) P_{jk}^B(t) c_k dt, \end{aligned} \tag{4.1}$$

where $P_{ik}^A(t)$ and $P_{jk}^B(t)$ are the probabilities of the A and B molecules to diffuse to voxel $k$ at time $t$, given that they are restricted to their yellow regions. $c_k$ is the reaction rate in voxel $k$.

The next question is: how to compute $P_{ik}^A(t)$ and $P_{jk}^B(t)$?

### 4.2.2 The DFSP algorithm

The computation of $P_{ik}^A(t)$ and $P_{jk}^B(t)$ requires some of the results from [25], the paper which introduces the Diffusive Finite State Projection (DFSP) method. Thus we will briefly introduce the DFSP algorithm.

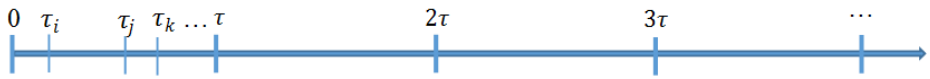



Figure 4.2: Time line of the simulation.

The purpose of DFSP is to avoid tracking the diffusion events. To reach this goal, the algorithm splits the reaction and diffusion processes in each step, and simulates them in turn. Figure 4.2 illustrates the time line for DFSP. It advances the system with a stepsize $\tau$. In each step, e.g. the first step, it does the following:

(i) Simulate the reaction process. e.g. using SSA to simulate the system up to time $\tau$ with no diffusion events. The $\tau_i$, $\tau_j$, $\tau_k$ in Figure 4.2 indicate the reaction event times.

(ii) At the end of the step, i.e. at time $\tau$, sample a diffusion process to redistribute the molecules in each voxel to their FSPs using the transition matrices.

Computing the transition matrix is expensive. Thus DFSP assumes that a molecule can diffuse only in a finite region in one step, which is called the finite state projection (FSP) in the algorithm. This name has previously appeared in [33] for solving chemical master equations. DFSP needs only to compute the transition matrix over the FSP. i.e. the probabilities $p_{ij}^A$ that an A molecule diffuses from voxel $i$ to voxel $j$ at time $\tau$, where $\tau$ is the simulation stepsize and voxel $j$ is inside the FSP (the yellow region in Figure 4.1a). The probabilities are normalized so that $p_{ij}^A$ sum up to one over the voxels

in the FSP.

$$\sum_{j \in \text{yellow region}} p_{ij}^{A} = 1.$$

DFSP computes the transition matrix by solving the diffusion master equation over the FSP.

TDPD uses a similar simulation procedure as DFSP, except that it differs from DFSP in two points. The first is that TDPD does not completely split the reaction and diffusion processes. In DFSP, reactants in different voxels cannot react in one step, even though they may be in neighboring voxels. In TDPD, diffusion is taken into account when sampling the reaction process. Thus, reactants originating in different voxels have a chance to react with each other in one step, as long as they diffuse to the same voxel. The other difference between TDPD and DFSP is that TDPD allows a molecule to diffuse outside its FSP in one step, which is reasonable since a molecule has a positive probability to diffuse anywhere in one step, even though the probabilities could be small for far away voxels. Diffusing out of the FSP is forbidden in DFSP.

The first difference implies that TDPD should use NSM [21] to sample the reaction events rather than SSA in step (i). This is because in DFSP, voxels are isolated in step (i). Thus DFSP can use SSA to simulate the reaction process for each voxel independently. However, in TDPD the voxels are coupled due to the consideration of diffusion. Thus the order of the events matters since one event may change the propensities of several voxels. Using NSM enables the algorithm to effectively simulate the events according to their time order.

The second difference implies that TDPD could have both reaction events and "diffusion events" in step (i). A diffusion event indicates that a molecule passes the boundary of its FSP. This never happens in DFSP since it does not allow a molecule to diffuse out of its FSP in one step.

Now we can outline the TDPD procedure in a similar way as for the DFSP. As shown in Figure 4.2, TDPD runs the simulation with a stepsize $\tau$. In each step, e.g. the first step, it does the following:

(I) Simulate the reaction and "diffusion" events using NSM. Here a "diffusion" event means that a molecule diffuses out of its FSP.

(II) At the end of the step, i.e. at time $\tau$, sample a diffusion process to redistribute the molecules in each voxel to their FSPs, using the transition matrices.

Here the diffusion process in step (II) is different from the "diffusion" events in step (I). In a "diffusion" event in step (I), a molecule will escape its FSP. However in the diffusion process in step (II), a molecule will diffuse to a voxel inside its FSP.

### 4.2.3 Time dependent transition matrix on an unstructured mesh

Next we return to the question of computing $P_{ik}^A(t)$ and $P_{jk}^B(t)$. Applying the matrix exponential technique from DFSP, we can compute the transition matrix $p_{ik}^A$ with stepsize $\tau$. However, here we need $P_{ik}^A(t)$ for any time $t \leq \tau$. A natural way to compute this is by linear interpolation:

$$P_{ik}^A(t) = \frac{p_{ik}^A}{\tau} t = r_{ik}^A t. \quad k \neq i. \tag{4.2}$$

Here $r_{ik}^A = p_{ik}^A/\tau$ can be considered to be the probability change rate. In the case of $k = i$, the probability $P_{ii}^A(t)$ is actually decreasing over time. The interpolation for this voxel is formulated as

$$P_{ii}^A(t) = 1 - \frac{1 - p_{ii}^A}{\tau} t = 1 + r_{ii}^A t. \tag{4.3}$$

We note that $r_{ii}^A = -\left(1 - p_{ii}^A\right)/\tau$ is also a probability change rate, which is negative in this case.

Combining the probability $p_{ik}^A$ from the DFSP and the interpolation formulas (4.2) and (4.3), we obtain the transition matrix for any species originating in any voxel.

As mentioned in subsection 4.2.2, one of the differences between TDPD and DFSP is that TDPD considers the effect of diffusion when simulating the reaction process. Thus, reactants originating in different voxels may contribute to the propensity function as well. In the next subsection, we demonstrate how the propensity function is constructed in TDPD.

### 4.2.4 Computing the time dependent propensity

As mentioned in step (I) in section 4.2.2, TDPD uses NSM to simulate the reaction and "diffusion" events in each step. Thus we must compute the time dependent propensity and then generate the next event time, for each voxel. In this subsection we show how to compute the reaction propensity $a_i(t)$ for a given voxel $i$. The diffusion propensity will be developed in the next subsection.

In the original NSM algorithm in [21], only reactants in the same voxel can contribute propensity in each step. However, in a TDPD simulation step, reactants originating in different voxels may also contribute propensity, as shown in Equation (4.1) for the example in Figure 4.1b. Thus in TDPD, we count the propensity $a_i(t)$ for voxel $i$ as follows: for a reaction event whose reactants all originate in voxel $i$, its propensity contributes to $a_i(t)$. If one of the reactants originates in voxel $i$ and the other reactant originates in voxel $j$, half of its propensity contributes to $a_i(t)$. The other half contributes to $a_j(t)$, the propensity for voxel $j$.

Let us continue with the example in Figure 4.1b. Equation (4.1) gives the time dependent propensity $a_{ij}(t)$ for a single pair of A and B molecules. Here $P_{ik}^A(t)$ and $P_{jk}^B(t)$ use the linear interpolation (4.2) or (4.3). Thus $a_{ij}(t)$ is a polynomial in $t$ of up to second order.

Suppose there are $X_i^A(0)$ A molecules in voxel $i$, and $X_j^B(0)$ B molecules in voxel $j$ at time 0. Then the total propensity of these molecules at time $t \leq \tau$ is $X_i^A(0)X_j^B(0)a_{ij}(t)$, since there are $X_i^A(0)X_j^B(0)$ A–B molecule pairs.

Since $X_i^A(0)X_j^B(0)a_{ij}(t)$ is contributed to by reactants from both voxel $i$ and voxel $j$, only half of its value, i.e.

$$a_{ij}^{A,B}(t) = \frac{1}{2}X_i^A(0)X_j^B(0)a_{ij}(t), \tag{4.4}$$

is counted in $a_i(t)$. Here the order of the superscript matters. $a_{ij}^{A,B}$ means A originates in voxel $i$ and B originates in voxel $j$, while $a_{ij}^{B,A}$ means B originates in voxel $i$ and A originates in voxel $j$.

Summing up $a_{ij}^{A,B}(t)$ over $j$ yields the total propensity $a_i^A(t)$ that species A, originating in voxel $i$, contributed to $a_i(t)$,

$$a_i^A(t) = \sum_{j=1}^{N} a_{ij}^{A,B}(t), \tag{4.5}$$

where $N$ is the total number of voxels. In simulation we do not need to go over $j$ from 1 to $N$. We need only add up the voxels whose FSP of species B overlap with the FSP of species A originating in voxel $i$. Similarly, the propensity contributed to $a_i(t)$ by the B molecules originating in voxel $i$ is given by

$$a_i^B(t) = \sum_{j=1}^{N} a_{ij}^{B,A}(t). \tag{4.6}$$

Since we have only one reaction channel $A+B \to \phi$ in the system, the reaction propensity $a_i(t)$ is the sum of $a_i^A(t)$ and $a_i^B(t)$,

$$a_i(t) = a_i^A(t) + a_i^B(t). \tag{4.7}$$

If A and B are the same species, i.e. the reaction is actually $A + A \to \phi$, Equation (4.7) becomes

$$a_i(t) = a_i^A(t). \tag{4.8}$$

The term $a_{ii}^{A,A}(t)$ in this case has a different form from (4.4):

$$a_{ii}^{A,A}(t) = \frac{1}{2} X_i^A(0)(X_i^A(0) - 1)a_{ii}(t), \tag{4.9}$$

because if both of the two A molecules originate in voxel $i$, there are $X_i^A(0)(X_i^A(0)-1)/2$ possible pairs.

Now we have obtained the reaction propensity $a_i(t)$ for voxel $i$. In order to compute the total propensity of voxel $i$, as in the NSM, we also need to compute the diffusion propensity for voxel $i$.

### 4.2.5 Diffusion events

As mentioned in subsection 4.2.2, a difference between TDPD and DFSP is that TDPD allows a molecule to diffuse out of its FSP in one step, which triggers a "diffusion event" in the simulation. This feature compensates the cutoff error of DFSP, thus it makes it possible to use a small FSP with larger stepsize when necessary. In NSM, a molecule is associated with a diffusion coefficient $\kappa$, where $\kappa dt$ gives the probability that the molecule jumps out of the voxel in the infinitesimal $dt$. We approximate the diffusion events in TDPD in the same way. We define a "diffusion coefficient" $\kappa_i^A$ for species A in voxel $i$, where $\kappa_i^A dt$ denotes the probability that an A molecule diffuses out of its FSP in the next infinitesimal $dt$. In this subsection, we describe how $\kappa_i^A$ is estimated.

A simple modification of DFSP enables it to compute the probability $\epsilon_i^A$ that an A molecule which originates in voxel $i$ appears outside of its FSP after a stepsize $\tau$ (which is called "error" in DFSP):

$$\epsilon_i^A = 1 - \mathbf{P}(\text{an A molecule which originates in voxel } i \text{ stays in its FSP at time } \tau).$$

Since we assume that the A molecule has a "diffusion propensity" $\kappa_i^A$, it follows that the probability that the A molecule remains in its FSP after a stepsize $\tau$ is

$\exp(-\kappa_i^A \tau)$, which follows an exponential distribution. Taking

$$\epsilon_i^A \approx 1 - \exp(-\kappa_i^A \tau),$$

we can estimate $\kappa_i^A$ as

$$\kappa_i^A \approx -\frac{\ln\left(1-\epsilon_i^A\right)}{\tau}, \tag{4.10}$$

and the diffusion propensity of species A originating in voxel $i$ at time $t$ is $X_i^A(t)\kappa_i^A$. With this "diffusion propensity", we can simulate the "diffusion events" as we do in the NSM.

Now we have computed both the reaction propensity and diffusion propensity for voxel $i$. Similar to the procedure used in NSM, we sum them up to obtain the total propensity, and then sample the next event time for the voxel.

### 4.2.6 Sample the next event time

Combining the reaction and diffusion propensities described in the previous two subsections, the total propensity of voxel $i$ is given by

$$a_{0,i}(t) = a_i(t) + X_i^A(t)\kappa_i^A + X_i^B(t)\kappa_i^B. \tag{4.11}$$

Applying Equation (3.3) in Chapter 3, the time to the next event $\tau_i$ for voxel $i$ can be obtained by solving

$$-\ln r_i = \int_0^{\tau_i} a_{0,i}(s)ds, \tag{4.12}$$

where $r_i$ is a uniform random number in $(0, 1)$.

For every voxel we can compute the next event time $\tau_i, \quad i = 1, \ldots, N$. This enables us to use a priority queue to find the voxel with the smallest $\tau_i$, as is done in the NSM. After the voxel with the smallest $\tau_i$ has been found, the next step is to sample an event for the voxel.

### 4.2.7 Sample the next event

Picking the first element from the priority queue gives us the voxel $i$ with the smallest $\tau_i$. The next event should occur at time $\tau_i$ due to at least one of the molecules originating in voxel $i$. According to Equation (4.11), the next event could be a reaction event with probability $a_i(\tau_i)/a_{0,i}(\tau_i)$, or a diffusion event of species A with probability $X_i^A(t)\kappa_i^A/a_{0,i}(\tau_i)$, or a diffusion event of species B with probability $X_i^B(t)\kappa_i^B/a_{0,i}(\tau_i)$.

#### 4.2.7.1 Reaction event

If the event is a reaction event, we must sample where the two reactant molecules originate and where the reaction event occurs. Equation (4.7) shows that the reaction propensity $a_i(t)$ is the sum of $a_i^A(t)$ and $a_i^B(t)$, where $a_i^A(t)$ is the reaction propensity that species A originating in voxel $i$ contributed to $a_i(t)$, and similarly for $a_i^B(t)$. Thus the probability that the reactant A (B) of the event originates in voxel $i$ is

$$\frac{a_i^A(t)}{a_i(t)}, \quad \left(\frac{a_i^B(t)}{a_i(t)}\right) \tag{4.13}$$

respectively.

Without loss of generality, assume that the reactant A originates in voxel $i$. The next step is to determine where the other reactant molecule originates. According to Equation (4.5), the probability that the B molecule originates in voxel $j$ is

$$a_{ij}^{A,B}(t)/a_i^A(t). \tag{4.14}$$

To simplify the computation, we need only to search over the voxels whose FSP of species B overlap with the FSP of species A originating in voxel $i$.

Suppose that the B molecule is selected from voxel $j$. Then the last step is to sample where the reaction event occurs. From Equation (4.1), the reaction event occurs in voxel $k$ with probability

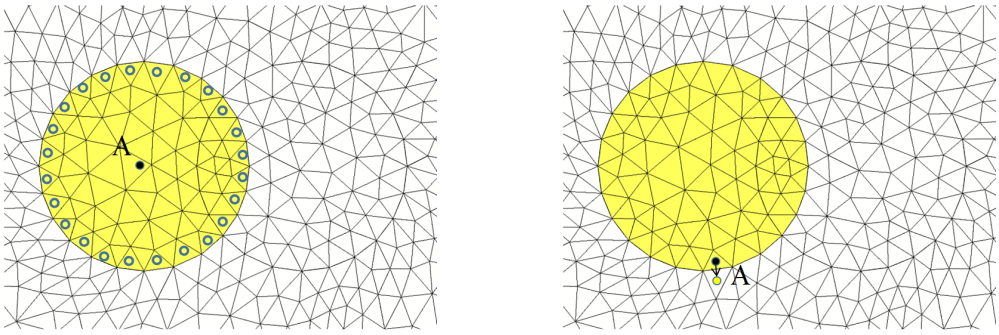
$$P_{ik}^A(t)P_{jk}^B(t)c_k/a_{ij}(t), \tag{4.15}$$

where voxel $k$ is in the overlapped region of the two molecules' FSPs.

The previous procedure completes the reaction event sampling. The next subsection describes how to sample a diffusion event.

#### 4.2.7.2 Diffusion event

If the event is a diffusion event, we need to determine where the molecule transfers to. Suppose that an A molecule triggers a diffusion event at time $\tau_i$. This molecule should be in one of the boundary voxels of its FSP at $\tau_i$, as shown by the voxels with empty dots in Figure 4.3a, and is about to jump out of the FSP. Let $B$ be the set of voxels with empty dots. The probability that the molecule is in a boundary




(a) Boundary voxels of the FSP (the voxels with empty dots.)

(b) Jump from a boundary voxel to a voxel outside the FSP.

Figure 4.3: Demonstration of a diffusion event

voxel $j$ is given by

$$\frac{P_{ij}^A(\tau_i)}{\sum_{k\in B} P_{ik}^A(\tau_i)},$$

where $P_{ij}^A(\tau_i)$ is given by (4.2) or (4.3).

Suppose that the A molecule is in voxel $j$ at time $\tau_i$ (the voxel with a solid black dot in Figure 4.3b). Next we pick an outbound direction for the molecule. Here we use the diffusion coefficient between adjacent voxels (which is also used in NSM for computing diffusion propensities). Let $O$ be the set of voxels adjacent to voxel $j$ and outside the FSP. For any voxel $l$ adjacent to voxel $j$ and outside the FSP, the probability that the A molecule jumps to it is given by

$$\frac{D_{jl}^A}{\sum_{k\in O} D_{jk}^A},$$

where $D_{jl}^A$ is the diffusion coefficient of species A from voxel $j$ to voxel $l$.

Suppose the voxel with the yellow dot in Figure 4.3b is the destination that is sampled for the diffusion event. We update the system state and finish the diffusion step.

### 4.2.8 Update system state

After sampling an event, we need to update the system state. We must update not only the species population in each voxel, but also the time dependent propensities.

#### 4.2.8.1 Updating the propensity when the species population increases

Suppose an event at time $\tau_i$ added $n$ A molecules to voxel $i$. These molecules can react with B molecules in the future. Thus to update the propensity, we need to add the propensity contributed by the new A molecules to $a_i(t)$.

Similarly to Equation (4.1), as shown in Figure 4.1b, the propensity that the new A molecules react in an orange voxel $k$ with B molecules originating in voxel $j$ at time $\tau_i \leq t \leq \tau$ is given by

$$\Delta a^A_{ijk}(t) = nX^B_j(t)P^A_{ik}(t-\tau_i)P^B_{jk}(t)c_k. \tag{4.16}$$

Here, $P^A_{ik}(t-\tau_i)$ implies that a new A molecule added to voxel $i$ at time $\tau_i$ begins its diffusion process from time $\tau_i$.

In Equation (4.16), we assume that no event occurring during $[0, \tau_i]$ places B molecules in voxel $j$. However, if there exists an event that places $m$ B molecules in voxel $j$ at some time $s \leq \tau_i$, i.e. $X^B_j(t) = X^B_j(0) + m$ for $s \leq t \leq \tau_i$, the propensity of

the new A molecules to react in an orange voxel $k$ with the B molecules originating in voxel $j$ is given by

$$\begin{aligned}\Delta a^A_{ijk}(t) &= \left(nX^B_j(0)P^A_{ik}(t-\tau_i)P^B_{jk}(t) + nmP^A_{ik}(t-\tau_i)P^B_{jk}(t-s)\right)c_k \\ &= nP^A_{ik}(t-\tau_i)\left(X^B_j(0)P^B_{jk}(t) + mP^B_{jk}(t-s)\right)c_k \\ &= nP^A_{ik}(t-\tau_i)\left(X^B_j(t)P^B_{jk}(t) - msr^B_{jk}\right)c_k. \end{aligned} \tag{4.17}$$

Here the scecond equality uses Equation (4.2) or (4.3). Comparing Equation (4.16) and (4.17), we can see that (4.17) has an extra term $-msr^B_{jk}$ (highlighted with green color in (4.17)). Actually this result can be extended to the case of several events. If there are $l$ events adding $m_1, \ldots, m_l$ B molecules to voxel $j$ at times $s_1, \ldots, s_l$, the extra term turns out to be -$\sum_{u=1}^{l} m_u s_u r^B_{jk}$. Thus in the simulation, we can define a variable $d^B_j$ that is initialized to 0. If an event occurs at time $s$ that places $m$ B molecules in voxel $j$, we update $d^B_j$ by

$$d^B_j \leftarrow d^B_j + ms. \tag{4.18}$$

Thus $\Delta a^A_{ijk}(t)$ can be represented by

$$\Delta a^A_{ijk}(t) = nP^A_{ik}(t-\tau_i)\left(X^B_j(t)P^B_{jk}(t) - d^B_j(t)r^B_{jk}\right)c_k. \tag{4.19}$$

Equation (4.19) gives the propensity that the new A molecules and the B molecules originating in voxel $j$ react in an orange voxel $k$. Summing $\Delta a^A_{ijk}(t)$ over $k$ for the orange region yields the total propensity for the new A molecules to react with

the B molecules originating in voxel $j$. Using (4.2) and (4.3), we obtain

$$\Delta a_{ij}^{A}(t) = \sum_{k\in\text{orange region}} \Delta a_{ijk}^{A}(t) = n(t-\tau_i)\left(X_j^B(t)t - d_j^B(t)\right) \sum_{k\in\text{orange region}} r_{ik}^A r_{jk}^B c_k$$
$$+ nP_{ij}^A(t-\tau_i)X_j^B(t)c_j + \begin{cases} n\left(X_j^B(t)P_{ji}^B(t) - d_j^B(t)r_{ji}^B\right)c_i & j\neq i \\ n\left(X_j^B(t)t - d_j^B(t)\right)r_{ji}^B c_i & j=i. \end{cases} \tag{4.20}$$

In the case of "B=A", the $n$ new A molecules can react with each other as well, thus the $\Delta a_{iik}^A(t)$ should have an extra term (highlighted with green color),

$$\Delta a_{iik}^{A}(t) = nP_{ik}^A(t-\tau_i)\left(X_i^A(t)P_{ik}^A(t) - d_i^A(t)r_{ik}^A\right)c_k + \frac{1}{2}n(n-1)\left(P_{ik}^A(t-\tau_i)\right)^2 c_k,$$

and $\Delta a_{ii}^A(t)$ becomes

$$\Delta a_{ii}^{A}(t) = n(t-\tau_i)\left(X_i^A(t)t - d_i^A(t)\right)\sum_k \left(r_{ik}^A\right)^2 c_k$$
$$+ nP_{ii}^A(t-\tau_i)X_i^A(t)c_i + n\left(X_i^A(t)t - d_i^A(t)\right)r_{ii}^A c_i$$
$$+ \frac{1}{2}n(n-1)(t-\tau_i)^2\sum_k \left(r_{ik}^A\right)^2 c_k + \frac{1}{2}n(n-1)\left(1+2(t-\tau_i)r_{ii}^A\right)c_i. \tag{4.21}$$

We note that $\Delta a_{ij}^A(t)$ is the propensity for the new A molecules to react with the B molecules originating in voxel $j$. To update the system propensity, $\Delta a_{ij}^A(t)$ should be added to $a_{ij}^{A,B}(t)$, which is the propensity contributed by the "old" A molecules originating in voxel $i$ and the B molecules originating in voxel $j$.

$$a_{ij}^{A,B}(t) \leftarrow a_{ij}^{A,B}(t) + \Delta a_{ij}^A(t). \tag{4.22}$$

After updating $a_{ij}^{A,B}(t)$, we update $a_i^A(t)$, $a_i(t)$ and $a_{0,i}(t)$ using (4.5), (4.7) or (4.8) and (4.11) respectively.

#### 4.2.8.2 Updating the propensity when the species population decreases

Suppose an event which occurs at time $\tau_i \leq \tau$ decreases the population of species A originating from voxel $i$ by $n$, i.e. $X_i^A(\tau_i+) = X_i^A(\tau_i-) - n$. To update the system propensity, we need to compute the propensity loss due to the loss of the $n$ A molecules.

According to Equation (4.4), the propensity contributed by the A molecules originating in voxel $i$ and the B molecules originating in voxel $j$ is $a_{ij}^{A,B}(t) + a_{ji}^{B,A}(t)$. This propensity loses part of its value $\Delta a_{ij}^A(t)$ after the event:

$$\Delta a_{ij}^A(t) = -\frac{n}{X_i^A(\tau_i-)} \left( a_{ij}^{A,B}(t) + a_{ji}^{B,A}(t) \right). \tag{4.23}$$

Since $a_{ij}^{A,B}(t) + a_{ji}^{B,A}(t)$ is the propensity without consideration of the event, (4.23) implies that the propensity loss is proportional to the ratio $n/X_i^A(\tau_i-)$, which is the ratio of molecules lost. (4.23) is exact if no event occurring before $\tau_i$ places A molecules into voxel $i$. However if some of the A molecules are generated by earlier events, Equation (4.23) is only an approximation. For example, suppose that one event occurs earlier than $\tau_i$, and generates $m$ A molecules in voxel $i$ at time $s < \tau_i$, i.e. $X_i^A(t) = X_i^A(0) + m$ for $s \leq t < \tau_i$, and then a new event at time $\tau_i$ consumes $n$ A molecules. We cannot tell how many of the $n$ molecules came from the original $X_i^A(0)$ molecules and how many of them came from the $m$ molecules that were generated later. Equation (4.23) in this case specifies that we divide it proportionally, i.e. $n \cdot X_i^A(0)/(X_i^A(0) + m)$ original molecules and $n \cdot m/(X_i^A(0) + m)$ generated molecules are consumed. This is

shown below:

$$n\frac{X_i^A(0)}{X_i^A(0)+m}P_{ik}^A(t)X_j^B(t)P_{jk}^B(t) + n\frac{m}{X_i^A(0)+m}P_{ik}^A(t-s)X_j^B(t)P_{jk}^B(t)$$
$$=\frac{n}{X_i^A(0)+m}\left(X_i^A(0)P_{ik}^A(t)+mP_{ik}^A(t-s)\right)X_j^B(t)P_{jk}^B(t), \tag{4.24}$$

where $\left(X_i^A(0)P_{ik}^A(t)+mP_{ik}^A(t-s)\right)X_j^B(t)P_{jk}^B(t)$ is the propensity without consideration of the event that consumes $n$ A molecules. In this example, $d_i^A$ is also changed in the event. According to Equation (4.18), $d_i^A = ms$ before the event at time $\tau_i$, where $m$ is the number of A molecules placed in voxel $i$ at time $s$. After the event, $n\cdot m/(X_i^A(0)+m)$ of the $m$ molecules are lost, thus the value of $d_i^A$ is updated by

$$d_i^A = \left(m - n\frac{m}{X_i^A(0)+m}\right)s = ms\left(1-\frac{n}{X_i^A(\tau_i-)}\right).$$

In other words, we update $d_i^A$ after the event as follows:

$$d_i^A \leftarrow d_i^A\left(1-\frac{n}{X_i^A(\tau_i-)}\right). \tag{4.25}$$

In the case of $B=A$ and $i=j$, both of the two reactant A molecules originate in voxel $i$. When $n$ $A$ molecules are removed, the number of A–A pairs changes from $X_i^A(\tau_i-)(X_i^A(\tau_i-)-1)/2$ to $X_i^A(\tau_i+)(X_i^A(\tau_i+)-1)/2$, where $X_i^A(\tau_i+) = X_i^A(\tau_i-)-n$. Thus we approximate $\Delta a_{ii}^A(t)$ by

$$\Delta a_{ii}^A(t) = -\left(1-\frac{X_i^A(\tau_i+)(X_i^A(\tau_i+)-1)}{X_i^A(\tau_i-)(X_i^A(\tau_i-)-1)}\right)a_{ii}^{A,A}(t). \tag{4.26}$$

Similarly to the procedure in the last subsection, to update the system propensity, $\Delta a_{ij}^A(t)$ is applied to $a_{ij}^{A,B}$ using (4.22). Then $a_i^A(t)$, $a_i(t)$ and $a_{0,i}(t)$ are updated using (4.5), (4.7) or (4.8) and (4.11) respectively.

Now we have finished the step of updating the propensity for a voxel. In the NSM, when the propensity of a voxel is updated, its next event time also needs to be updated. Here we use the same procedure. After updating the propensity of a voxel, we need to update its next event time.

#### 4.2.8.3 Update the next event time

After updating the event at time $\tau_i$ for voxel $i$, we need to sample a new time $\tau^{\text{new}}$ for the next event. Here we need to sample a new uniform random number $r_i$ in $(0, 1)$ and solve Equation (4.12) again. The only difference is that the integration begins from time $\tau_i$,

$$-\ln r_i = \int_{\tau_i}^{\tau^{\text{new}}} a_{0,i}(s)ds.$$

Sometimes an event may change the propensity of several voxels. For example, if the event at time $\tau_i$ is a diffusion event where an A molecule diffuses from voxel $i$ to voxel $k$, we must sample $\tau^{\text{new}}$ for voxel $i$, and also must update $\tau_k$ for voxel $k$, since a new molecule has arrived. Without loss of generality, suppose the event at $\tau_i$ is the first event that changes the propensity of voxel $k$, whose value is changed from $a_{0,k}(t)$ to $a_{0,k}^{\text{new}}(t)$. Originally, $\tau_k$ is computed by solving Equation (4.12). Now the integration must be updated for the time interval between $\tau_i$ and $\tau_k$, since its propensity is updated. Thus Equation (4.12) becomes

$$-\ln r_k = \int_0^{\tau_i} a_{0,k}(s)ds + \int_{\tau_i}^{\tau_k} a_{0,k}^{\text{new}}(s)ds. \tag{4.27}$$

Solving this equation yields the updated $\tau_k$. Here $a_{0,k}(s)$ and $a_{0,k}^{\text{new}}(s)$ are polynomials of $s$ of at most second order. Thus we use their integrals $F_k(s) = \int a_{0,k}(s)ds$ and $F_k^{\text{new}}(s) = \int a_{0,k}^{\text{new}}(s)ds$ in (4.27), yielding

$$-\ln r_k + F_k(0) - F_k(\tau_i) + F_k^{\text{new}}(\tau_i) = F_k^{\text{new}}(\tau_k).$$

The previous deduction can be extended to the situation of multiple events. If $l$ events occur at times $s_1, \ldots, s_l$ that update the propensity of voxel $k$ from $F_k(t)$ to $F_k^1(t), \ldots, F_k^l(t)$, then we can define a variable $f$ that begins with $f = -\ln r_k + F_k(0)$, and updates its value after each event by

$$f \leftarrow f - F_k^{u-1}(s_u) + F_k^u(s_u), \quad u = 1, \ldots, l.$$

Then the next event time $\tau_k$ can be updated by solving

$$f = F_k^l(\tau_k).$$

After updating the next event time, we update the priority queue. Now we are at time $\tau_i$ in the time line shown in Figure 4.2. We go on sampling the next event and repeat this procedure until we reach time $\tau$, the simulation stepsize. Then we are at time $\tau$ and have finished the step (I) in subsection 4.2.2. We are next going to outline step (II), i.e. sample a diffusion process that redistributes the molecules within their FSP.

### 4.2.9 Sampling the diffusion process at the end of a step

When the simulation time reaches the stepsize $\tau$, we must redistribute the molecules within their FSP due to the diffusion. This step is the same as the corresponding step in Chapter 3. The distribution of the initial A molecules originating in voxel $i$ follows a multinomial distribution. The probability that a molecule diffuses to voxel $j$ at time $\tau$, which is within the FSP, is $p_{ij}^A$ according to Equation (4.2) and (4.3).

For molecules generated by events during the step, the multinomial should have different parameters. For example, suppose that $n$ A molecules are generated in voxel $i$ at time $\tau_i$. The probability for one of these molecules to diffuse to voxel $j$ at time $\tau$ is $P_{ij}^A(\tau - \tau_i)$, as shown by Equation (4.2) or (4.3). To distribute the $n$ molecules at the end of the step we need to keep a record of $(\tau_i, n)$. This can be potentially expensive if the system has many events in a step.

In our code, we made a compromise. An approximation is used for the distribution which does not require the storage of the events' information. Following the example in the last paragraph, suppose that there are $X_i^A(t) = X_i^A(0) + n$ A molecules at time $\tau_i \leq t \leq \tau$. We approximate their distribution in the FSP by a multinomial distribution. The probability $Q_{ij}^A(t)$ that a molecule is distributed to voxel $j$ satisfies

$$\begin{aligned} X_i^A(t)Q_{ij}^A(t) &= X_i^A(0)P_{ij}^A(t) + nP_{ij}^A(t-\tau_i) \\ \implies Q_{ij}^A(t) &= P_{ij}^A(t) - \frac{1}{X_i^A(t)} r_{ij}^A d_i^A(t). \end{aligned} \tag{4.28}$$

This approximation conserves the expected number of molecules which are diffused to each voxel in the FSP.

In the simulation, it is more convenient to first sample the number of molecules staying in voxel $i$. The molecules that are not staying are then distributed to other voxels in the FSP. Since $P_{ij}^A(t)$ in (4.2) is linear with respect to $t$, the probability that an A molecule diffuses to voxel $j$, given that it does not stay in voxel $i$, is $p_{ij}^A/(1-p_{ii}^A)$. This is a convenient property since we no longer need to compute (4.28).

After redistributing the molecules, we have completed one step of the simulation. We go on simulating the next step from $\tau$ to $2\tau$, as shown in Figure 4.2. This procedure is repeated until the end time is reached. In the next subsection, we will summarize the procedure of the algorithm.

### 4.2.10 Summary of the algorithm

In this section we present the algorithm in a more general setting. Suppose the system has $M$ reactions $R_1, \ldots, R_M$, and $N$ species $S_1, \ldots, S_N$. Assume that the current state of the system is $\mathbf{X}$, and without loss of generality, the current time is 0. In the previous sections we have introduced the time dependent propensity $a_i(t)$ of second order reactions, which is given by (4.7) or (4.8). For zeroth and first order reactions, the time dependent propensity functions for voxel $i$ are

- $\phi \xrightarrow{c_i} something$

$$a_i(t) = c_i. \tag{4.29}$$

- $A \xrightarrow{c} something$

$$a_i(t) = cX_i^A(t). \tag{4.30}$$

The total propensity contributed by voxel $i$ is given by

$$a_{0,i}(t) = \sum_{l=1}^{M} a_i^l(t) + \sum_{r=1}^{N} X_i^r(t)\kappa_i^r, \tag{4.31}$$

where $a_i^l(t)$ is the propensity function of reaction $l$ at time $t$. $X_i^r(t)$ is the population of species $S_r$ originating from voxel $i$ at time $t$. Its diffusion propensity is $\kappa_i^r$, which is given by Equation (4.10).

The steps of the TDPD algorithm are listed below

0. Compute the transition matrix and diffusion propensities using DFSP. For every second order reaction, find the overlapped voxels of the FSPs (the orange voxels in Figure 4.1b). For each voxel, compute the zeroth order reaction propensities using (4.29) (These values need only be computed once).

   For each realization, do the following:

1. Initialize the time $t = t_0$ and the system state $\mathbf{X} = \mathbf{X}_0$.

2. For each voxel $i$ and second order reaction $A + B \rightarrow$ something, compute $a_{ij}^{A,B}(t)$ and $a_{ij}^{B,A}(t)$ using (4.4). Then compute $a_i^A(t)$ and $a_i^B(t)$ using (4.5) and (4.6). Finally compute $a_i(t)$ using (4.7). In the case of "B=A", Equation (4.9) and (4.8) should be used.

3. For each voxel $i$, set $d_i^A = 0$ for every species A. Compute the propensity for first order reactions using (4.30). Compute $a_{0,i}(t)$ using (4.31). Sample a uniform random number $r_i \in (0,1)$. Set $f_i = -\ln r_i + F_i(0)$. Here $F_i(t) = \int a_{0,i}(t)dt$. Generate the next event time $\tau_i$ by solving $f_i = F_i(\tau_i)$. Store the voxel indices in a priority queue according to their next event time.

4. Pick the first element in the priority queue to get the voxel $i$ for the next event. If $\tau_i > \tau$, go to step 11.

5. Sample the event type. According to Equation (4.31), the event could be firing a reaction $l$ with probability $a_i^l(\tau_i)/a_{0,i}(\tau_i), \quad l = 1, \ldots, M$, or it could be a diffusion event of species $S_r$ with probability $X_i^r(\tau_i)\kappa_i^r/a_{0,i}(\tau_i), \quad r = 1, \ldots, N$.

6. Sample where the reactant molecules come from and where the product is generated.

   - If in step 5 the sampled event is a reaction: $\phi \rightarrow$ something, the products are produced in voxel $i$.
   - If in step 5 the sampled event is a reaction: $A \rightarrow$ something, the reactant originates in voxel $i$. Suppose the product is produced in voxel $k$. Then $k$ is a random variable with point probability $Q_{ik}^A(\tau_i)$ given by Equation (4.28).
   - If in step 5 the sampled event is a reaction: $A + B \rightarrow$ something, according to Equation (4.13), the probability that reactant A originates in voxel $i$ is $a_i^A(t)/a_i(t)$. If reactant A originates in voxel $i$, sample the voxel $j$ from which

reactant B originates with probability $a_{ij}^{A,B}(t)/a_i^A(t)$, according to (4.14). Then sample the voxel $k$ where the reaction event occurs with probability

$$\frac{Q_{ik}^A(\tau_i)\,Q_{jk}^B(\tau_i)}{\sum_s Q_{is}^A(\tau_i)\,Q_{js}^B(\tau_i)}.$$

A similar procedure works for the case where reactant B originates in voxel $i$.

- If in step 5 the sampled event is a reaction: $A + A \to$ something, then one of the reactant molecules should originate in voxel $i$. Sample the voxel $j$ from which the other reactant originates with probability $a_{ij}^{A,A}(t)/a_i^A(t)$. Then sample the voxel $k$ where the reaction event occurs with probability

$$\frac{Q_{ik}^A(\tau_i)\,Q_{jk}^A(\tau_i)}{\sum_s Q_{is}^A(\tau_i)\,Q_{js}^A(\tau_i)}.$$

- If in step 5 the sampled event is a diffusion event of species A, as in Figure 4.3a, the A molecules should be in one of the empty-dot-voxels at time $\tau_i$, and about to jump outside the FSP. The probability that the molecule is in voxel $j$ is $Q_{ij}^A(\tau_i)/\sum_s Q_{is}^A(\tau_i)$. Here the sum is taken over the interior boundary subvolumes of the yellow region. Suppose voxel $j$ is selected (the voxel that has a solid black dot in Figure 4.3b). The molecule should jump to a voxel adjacent to voxel $j$ and outside the yellow region. The A molecule jumps to voxel $k$ (as shown by the voxel with a yellow dot in Figure 4.3b) with probability $D_{jk}/\sum_s D_{js}$. Here the sum is taken over the voxels adjacent to voxel $j$ and outside the FSP.

7. Update the system state. For every voxel $s$, if the event in step 6 changes the population of species A originating in it, do the following:

   - If the event increases the population of species A originating in voxel $s$ by $n$, then for any second order reaction $A + B \to$ something, compute $\Delta a^A_{s,j}(t)$ using (4.20) for every voxel $j$ whose FSP of species B overlaps with the FSP of species A originating in voxel $s$. In the case of "B=A", Equation (4.21) should be used. Update the following variables,

   $$d^A_s \leftarrow d^A_s + n\tau_i, \quad X^A_s \leftarrow X^A_s + n.$$

   - If the event decreases the population of species A originating in voxel $s$ by $n$, for any second order reaction $A + B \to$ something, compute $\Delta a^A_{s,j}(t)$ using (4.23) for every voxel $j$ whose FSP of species B overlaps with the FSP of species A originating in voxel $s$. In the case of "B=A", Equation (4.26) should be used. Update the following variables,

   $$d^A_s \leftarrow d^A_s(1 - n/X^A_s), \quad X^A_s \leftarrow X^A_s - n.$$

   - Update the following polynomials.

   $$a^{A,B}_{sj}(t) \leftarrow a^{A,B}_{sj}(t) + \Delta a^A_{s,j}(t)$$

   $$a^A_s(t) \leftarrow a^A_s(t) + \sum_j \Delta a^A_{s,j}(t)$$

   $$a_s(t) \leftarrow a_s(t) + \sum_j \Delta a^A_{s,j}(t)$$

   Here we need to store a copy of the old $a_s(t)$ for step 8.

- For any first order reaction $A \xrightarrow{c}$ something, update $a_s(t)$ using (4.30).

8. For any voxel $s$ whose propensity has changed, do the following:

   - Update $a_{0,s}(t)$ using (4.31).
   - Update $f_s$ as follows:
     - If voxel $s$ is the voxel chosen in step 4, sample a new random number $r_s \in (0,1)$. Set $f_s = -\ln r_s + F_s(\tau_i)$.
     - If voxel $s$ is not the voxel chosen in step 4, update $f_s$ by $f_s \leftarrow f_s - F_s^{\text{old}}(\tau_i) + F_s(\tau_i)$. Here the $F_s^{\text{old}}(t)$ is the integral of the old copy of $a_s(t)$ we saved in step 7.
   - Update the next event time by solving $f_s = F_s(\tau_s)$.

9. Update the priority queue.

10. Return to step 4.

11. Sample a diffusion process with stepsize $\tau$. For example, for species A in voxel $s$, sample a multinomial random variable to distribute the $X_s^A$ A molecules in its FSP. The probability that a molecule is distributed to voxel $j$ is $Q_{s,j}^A(\tau)$, given by (4.28). Repeat this procedure for each diffusive species in each voxel.

12. Advance $t \leftarrow t + \tau$. If $t < T$, return to Step 2, else stop the realization. Here $T$ is the total simulation time.

This algorithm has been implemented in the software package PyURDME [29].

## 4.3 Numerical simulation

In this section we present the simulation results generated by our new TDPD algorithm on a cylinder neutralization model.

### 4.3.1 Model description

In this example, we use the constructive solid geometry method [34] to create a spatial domain in the shape of a cylinder, as shown in Figure 4.4. The length dimension (x-axis) of the cylinder ranges from -5 to 5, and the circular dimension has diameter 1. Species A is created at the left circular edge, and B is created at the right circular edge. These two species diffuse through the volume of the cylinder and react to neutralize each other when they meet. Thus the reaction produces two neutral particles C. i.e. the reaction is $A + B \rightarrow 2C$. The time-averaged concentration of the species A and B should form a gradient away from their creation edge. The molecular creation rates are 10000 per unit volume per second, for both A and B. The neutralization reaction rate is 0.00001 unit volume per second. The diffusion rates for the three species are 0.1 unit length square per second. The simulation time was taken to be 200 seconds.

### 4.3.2 Simulation results

Figure 4.5a and 4.5b show the distribution of the two species over the x axis. Figure 4.5a was generated by TDPD with FSP size no greater than 100 voxels. Figure 4.5b was generated by NSM. The results are averaged over ten realizations. TDPD

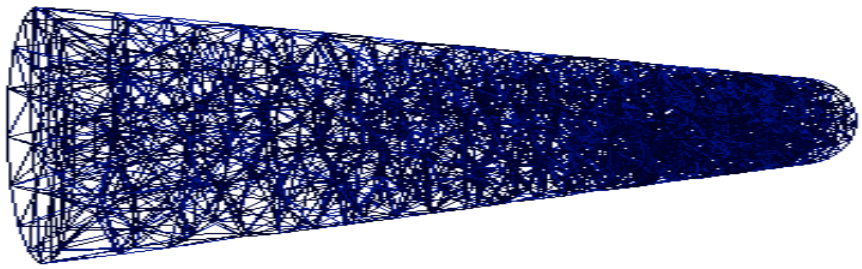
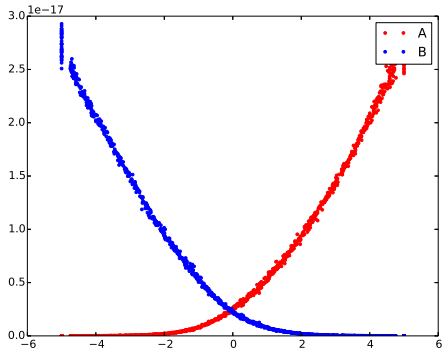
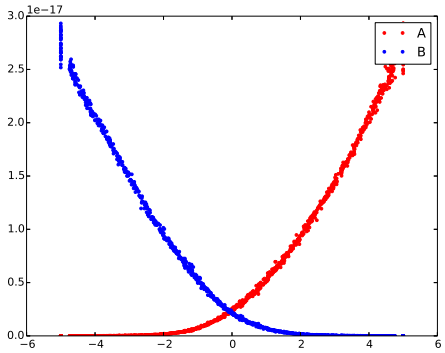


Figure 4.4: Geometry of the cylinder

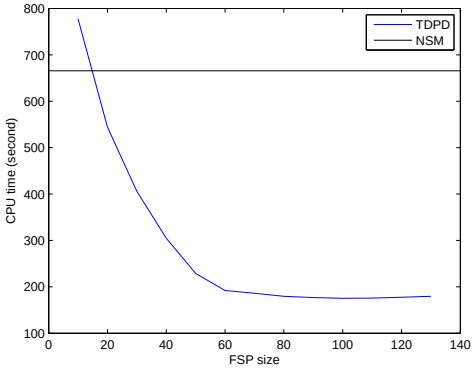
and NSM generate nearly identical results. Figure 4.5c shows the CPU times used by the TDPD with different FSP sizes. It can be seen that the TDPD runs faster as the FSP size increases. This is because a larger FSP size decreases the number of diffusion events. The horizontal line indicates the CPU time used by the NSM. TDPD has better performance than NSM when the FSP size is large. Figure 4.5d shows the CPU times used by TDPD and NSM with different molecular creation rates. Here the maximum FSP size of TDPD is 100. It shows that TDPD performance has a better slope than NSM with respect to the molecular creation rates. This is because the more molecules in the system, the more diffusion events for NSM. However for TDPD, the diffusion events can be greatly avoided by a proper selection of the FSP.



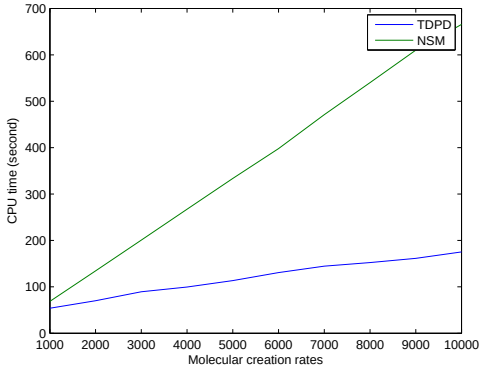

(a) Distribution of the two species over the $x$ axis. Value is averaged over ten TDPD realizations with maximum FSP size of 100.




(b) Distribution of the two species over the $x$ axis. Value is averaged over ten NSM realizations.




(c) CPU time used by TDPD with different FSP sizes. The horizontal line is the CPU time used by NSM.




(d) CPU time used by TDPD and NSM with different molecular creation rates. The maximum FSP size is 100.

Figure 4.5: Simulation results

## 4.4 Conclusion

TDPD on an unstructured mesh provides a means to accelerate the NSM for systems with many diffusive molecules. The key point of the method is that it uses the interpolation of the fixed stepsize transition matrix to compute the time dependent propensity. It also uses diffusion events to compensate the probability loss from the finite state projection, which helps the algorithm to maintain a good accuracy.

However, the method is not universally better than NSM. The computation of the time dependent propensity for reactants originating from two voxels is not easy. If there are only a few reactant pairs in the voxels, it may not be worth the cost to compute the time dependent propensity for them. NSM is cheaper in this case. TDPD can reduce the computation time only when there are many reactant pairs in the two voxels, where they can share the same time dependent propensity.

# Chapter 5

# Conclusion

## 5.1 Summary of the thesis

In this thesis we have developed algorithms to accelerate the stochastic simulation of chemical reaction systems. In Chapter 2, we showed how to use the time dependent solution to improve the performance of tau-leaping. In Chapter 3, we showed how to apply the time dependent propensity function to spatial stochastic simulation with a regular mesh in a rectangular domain, which yields a speed up over NSM for systems with many diffusion events. This idea was extended to unstructured mesh in Chapter 4, which enables it to simulate systems with complex geometries. We have implemented the algorithms in the software packages Stochkit 2 [19] and PyURDME [29].

## 5.2 Future directions

Figure 4.5c in Chapter 4 shows that the performance of TDPD depends on the FSP size. The algorithm could be more efficient if it could automatically pick the best FSP size for the simulation.

In Chapter 4 we interpolated the transition matrix linearly. Actually we could have used higher order interpolation. For example, suppose $P_{ij}(t)$ is the transition probability that a molecule diffuses from subvolume $i$ to subvolume $j$ at time $t$. We can use DFSP to compute its value at time $\tau$ and $\tau/2$. Then we can apply a second order interpolation to obtain

$$P_{ij}(t) = \begin{cases} 2\left(P_{ij}(\tau) - 2P_{ij}\left(\frac{\tau}{2}\right)\right)\frac{t^2}{\tau^2} + \left(4P_{ij}\left(\frac{\tau}{2}\right) - P_{ij}(\tau)\right)\frac{t}{\tau} & i \neq j \\ -2\left(2P_{ij}\left(\frac{\tau}{2}\right) - P_{ij}(\tau) - 1\right)\frac{t^2}{\tau^2} + \left(4P_{ij}\left(\frac{\tau}{2}\right) - P_{ij}(\tau) - 3\right)\frac{t}{\tau} + 1 & i = j \end{cases}.$$

# Bibliography


[1] Harley H. McAdams and Adam Arkin. Stochastic mechanisms in gene expression. *Proc. Natl. Acad. Sci. USA*, 94:814–819, 1997.

[2] Adam Arkin, John Ross, and Harley H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage $\lambda$-infected escherichia coli cells. *Genetics*, 149:1633–1648, 1998.

[3] Nina Fedoroff and Walter Fontana. Small numbers of big molecules. *Science*, 297:1129–1131, 2002.

[4] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81:2340–2361, 1977.

[5] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comput. Phys.*, 22:403–434, 1976.

[6] Yang Cao, Hong Li, and Linda R. Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *J. Chem. Phys.*, 121(9):4059–4067, 2004.

[7] Alexander Slepoy, Aidan P. Thompson, and Steven J. Plimpton. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. *J. Chem. Phys.*, 128:205101, 2008.

[8] Vo Hong Thanh, Corrado Priami, and Roberto Zunino. Efficient rejection-based simulation of biochemical reactions with stochastic noise and delays. *J. Chem. Phys.*, 141:134116, 2014.

[9] Michael A. Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104:1876–1889, 2000.

[10] Daniel T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 115(4):1716–1733, 2001.

[11] Yang. Cao and Linda. R. Petzold. Slow-scale tau-leaping method. *Comput. Methods Appl. Mech. Engrg.*, 197:3472–3479, 2008.

[12] Lee A. Segel and Marshall Slemrod. The quasi-steady-state assumption: a case study in perturbation. *SIAM Review*, 31:446–477, 1989.

[13] Christopher. V. Rao and Adam. P. Arkin. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the gillespie algorithm. *J. Chem. Phys.*, 118(11):4999–5010, 2003.

[14] Yang. Cao, Daniel. T. Gillespie, and Linda. R. Petzold. The slow-scale stochastic simulation algorithm. *J. Chem. Phys.*, 122:014116, 2005.

[15] Ethan A. Mastny, Eric L. Haseltine, and James B. Rawlings. Two classes of quasi-steady-state model reductions for stochastic kinetics. *J. Chem. Phys.*, 127:094106, 2007.

[16] Jin Fu, Sheng Wu, and Linda R. Petzold. Time dependent solution for acceleration of tau-leaping. *J. Comput. Phys.*, 235:446–457, 2013.

[17] Tobias Jahnke and Derya Altintan. Efficient simulation of discrete stochastic reaction systems with a splitting method. *BIT Numer. Math.*, 50:797–822, 2010.

[18] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold. Efficient step size selection for the tau-leaping simulation method. *J. Chem. Phys.*, 124:044109, 2006.

[19] Kevin R. Sanft, Sheng Wu, Min Roh, Jin Fu, Rone Kwei Lim, and Linda R. Petzold. Stochkit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics*, 27(17):2457–2458, 2011.

[20] C. W. Gardiner, K. J. McNeil, D. F. Walls, and I. S. Matheson. Correlations in stochastic theories of chemical reactions. *J. Stat. Phys.*, 14:307–331, 1976.

[21] J. Elf and M. Ehrenberg. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *IEE P. Syst. Biol.*, 1:230–236, 2004.

[22] Johan Hattne, David Fange, and Johan Elf. Stochastic reaction-diffusion simulation with mesord. *Bioinformatics*, 21:2923–2924, 2005.

[23] Brian Drawert, Stefan Engblom, and Andreas Hellander. A modular framework for stochastic simulation of reaction-transport processes in complex geometries. *BMC Syst. Biol.*, 6:76, 2012.

[24] Sotiria Lampoudi, Daniel T. Gillespie, and Linda R. Petzold. The multinomial simulation algorithm for discrete stochastic simulation of reaction-diffusion systems. *J. Chem. Phys.*, 130(9):094104, 2009.

[25] Brian Drawert, Michael J. Lawson, Linda R. Petzold, and Mustafa Khammash. The diffusive finite state projection algorithm for efficient simulation of the stochastic reaction-diffusion master equation. *J. Chem. Phys.*, 132(7):074101, 2010.

[26] Lars Ferm, Andreas Hellander, and Per Lötstedt. An adaptive algorithm for simulation of stochastic reaction-diffusion processes. Technical Report 2009-010, Department of Information Technology, Uppsala University, April 2009.

[27] Jin Fu, Sheng Wu, Hong Li, and Linda R. Petzold. The time dependent propensity function for acceleration of spatial stochastic simulation of reaction-diffusion systems. *J. Comput. Phys.*, 274:524–549, 2014.

[28] Mark Griffith, Tod Courtney, Jean Peccoud, and William H. S. Dynamic partitioning for hybrid simulation of the bistable HIV-1 transactivation network. *Bioinformatics*, 22:2782–2789, 2006.

[29] Brian Drawert and Andreas Hellander. Pyurdme github repository. `https://github.com/briandrawert/pyurdme/`.

[30] M. F. Hockin, K. C. Jones, S. J. Everse, and K. G. Mann. A model for the stoichiometric regulation of blood coagulation. *J. Biol. Chem.*, 277:18322–18333, 2002.

[31] Daniel T. Gillespie. *Markov Processes: An Introduction for Physical Scientists.* Academic Press, Inc., San Diego, 1992.

[32] Michael A. Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem.*, 104:1876–1889, 2000.

[33] Brian Munsky and Mustafa Khammash. The finite state projection algorithm for the solution of the chemical master equation. *J. Chem. Phys.*, 124:044104, 2006.

[34] James D. Foley. *Computer Graphics: Principles and Practice*, chapter 12.7, pages 557—558. Addison-Wesley Professional, 1996.

[35] Geoffery R. Grimmett and David R. Stirzaker. *Probabiliy and Random Processes*, chapter 5, page 154. Oxford University Press Inc., New York, third edition, 2001.

# Appendix

## A.1 Derivation of the time dependent solution

We use the probability generating function to derive the formula. For a non-negative discrete random variable $X$, its probability generating function is defined as

$$G_X(s) = \sum_{i=0}^{\infty} s^i p(X = i),$$

where $p(X = i)$ is the probability that $X$ takes the value of $i$. The generating function of a Poisson random variable $X \sim \mathcal{P}(\lambda)$ is given by

$$G_X(s) = \sum_{i=0}^{\infty} s^i p(X = i) = \sum_{i=0}^{\infty} s^i \frac{\lambda^i}{i!} e^{-\lambda} = e^{-\lambda} \sum_{i=0}^{\infty} \frac{(s\lambda)^i}{i!} = e^{-\lambda} e^{s\lambda} = e^{(s-1)\lambda}. \quad \text{(A.1.1)}$$

The joint generating function of multiple random variables $(X_1, \ldots, X_n)$ is defined as

$$G_{X_1,\ldots,X_n}(s_1, \ldots, s_n) = \sum_{i_1,\ldots,i_n} s_1^{i_1} \ldots s_n^{i_n} p(X_1 = i_1, \ldots, X_n = i_n). \quad \text{(A.1.2)}$$

It is convenient to compute the generating function of every variable from their joint generating function. For example, if we want the generating function of $X_j$, we can

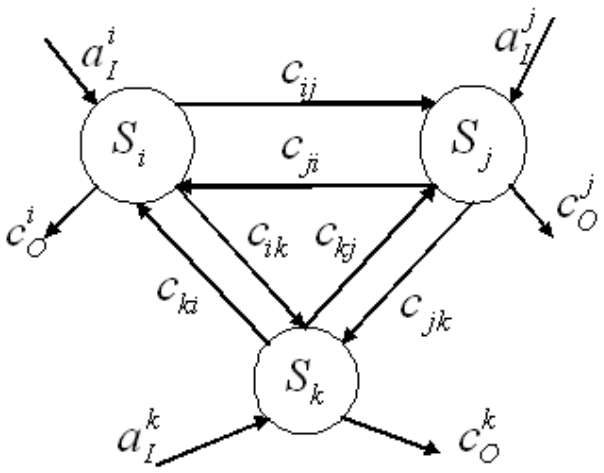



Figure A.1.1: Example system

simply plug $s_i = 1,\ i \neq j$ into (A.1.2). This is because

$$G_{X_1,\ldots,X_n}(1,\ldots,s_j,\ldots,1) = \sum_{i_1,\ldots,i_n} s_j^{i_j} p(X_1 = i_1,\ldots,X_n = i_n)$$

$$= \sum_{i_j} s_j^{i_j} \sum_{i_k,k\neq j} p(X_1 = i_1,\ldots,X_n = i_n) = \sum_{i_j} s_j^{i_j} p(X_j = i_j) = G_{X_j}(s_j). \qquad \text{(A.1.3)}$$

A useful property of the joint generating function is given by

Theorem 1: Random variables $(X_1,\ldots,X_n)$ are independent if and only if

$$G_{X_1,\ldots,X_n}(s_1,\ldots,s_n) = \prod_{i=1}^{n} G_{X_i}(s_i).$$

The proof can be found in any probability textbook (see Theorem (29) for two variable case in [35]).

Now let us look at the time dependent population of multiple species. Suppose that we have $n$ species $\hat{S} = \{S_1,\ldots,S_n\}$. As shown in Figure A.1.1, for each $S_i$ there

is an input from outside the system that increases the population of $S_i$ with propensity $a_I^i$, i.e. a reaction $R_I^i: \ \phi \to S_i$. There is also an output from $S_i$ with rate constant $c_O^i$, i.e. a reaction $R_O^i: \ S_i \to \phi$. In addition, one $S_i$ molecule can become a $S_j$ molecule due to a reaction $R_{ij}: \ S_i \to S_j$ with rate constant $c_{ij}$.

Denote by $x_i$ the population of species $S_i$, $r_O^i$ the number of firings of reaction $R_O^i$, $r_I^i$ the number of firings of reaction $R_I^i$, and $p_{i_1,\ldots,i_n,j_1,\ldots,j_n}(t)$ the probability that $x_1 = i_1, \ldots, x_n = i_n, \ r_O^1 = j_1, \ldots, r_O^n = j_n$. Then the master equation can be written as

$$\begin{aligned}
\frac{dp_{i_1,\ldots,i_n,j_1,\ldots,j_n}(t)}{dt} &= \sum_{k=1}^{n} p_{i_1,\ldots,i_k-1,\ldots,i_n,j_1,\ldots,j_n}(t) a_I^k \\
&+ \sum_{k\neq l} p_{i_1,\ldots,i_k+1,\ldots,i_l-1,\ldots,i_n,j_1,\ldots,j_n}(t) c_{kl}(i_k+1) + \sum_{k=1}^{n} p_{i_1,\ldots,i_k+1,\ldots,i_n,j_1,\ldots,j_k-1,\ldots,j_n}(t) c_O^k (i_k+1) \\
&- p_{i_1,\ldots,i_n,j_1,\ldots,j_n}(t) \left( \sum_{k=1}^{n} a_I^k + \sum_{k\neq l} c_{kl} i_k + \sum_{k=1}^{n} c_O^k i_k \right). \qquad \text{(A.1.4)}
\end{aligned}$$

To simplify the notation we will use $p_{i_k+1,j_l-1}$ to refer to $p_{i_1,\ldots,i_k+1,\ldots,i_n,j_1,\ldots,j_l-1,\ldots,j_n}$. Multiplying $s_1^{i_1}\ldots s_n^{i_n} u_1^{j_1}\ldots u_n^{j_n}$ on both sides of the master equation (A.1.4) gives

$$\begin{aligned}
\frac{\partial s_1^{i_1}\ldots s_n^{i_n} u_1^{j_1}\ldots u_n^{j_n} p_{i_1,\ldots,i_n,j_1,\ldots,j_n}(t)}{\partial t} &= \sum_{k=1}^{n} s_1^{i_1}\ldots s_k^{i_k-1}\ldots s_n^{i_n} u_1^{j_1}\ldots u_n^{j_n} p_{i_k-1} s_k a_I^k \\
&+ \sum_{k\neq l} c_{kl} s_l \frac{\partial}{\partial s_k}\left(\ldots s_k^{i_k+1}\ldots s_l^{i_l-1}\ldots p_{i_k+1,i_l-1}\right) + \sum_{k=1}^{n} c_O^k u_k \frac{\partial}{\partial s_k}\left(\ldots s_k^{i_k+1}\ldots u_k^{j_k-1}\ldots p_{i_k+1,j_k-1}\right) \\
&- s_1^{i_1}\ldots s_n^{i_n} u_1^{j_1}\ldots u_n^{j_n} p_{i_1,\ldots,i_n,j_1,\ldots,j_n} \sum_{k=1}^{n} a_I^k - \sum_{k\neq l} c_{kl} s_k \frac{\partial}{\partial s_k}\left(s_1^{i_1}\ldots s_n^{i_n} u_1^{j_1}\ldots u_n^{j_n} p_{i_1,\ldots,i_n,j_1,\ldots,j_n}\right) \\
&- \sum_{k=1}^{n} c_O^k s_k \frac{\partial}{\partial s_k}\left(s_1^{i_1}\ldots s_n^{i_n} u_1^{j_1}\ldots u_n^{j_n} p_{i_1,\ldots,i_n,j_1,\ldots,j_n}\right).
\end{aligned}$$

Summing both sides over $i_1, \ldots, i_n, j_1, \ldots, j_n$ and using the definition of generating function (A.1.2), we have

$$\begin{aligned}
&\frac{\partial G\left(s_1, \ldots, s_n, u_1, \ldots, u_n, t\right)}{\partial t} \\
&= \sum_{k=1}^{n} G s_k a_I^k + \sum_{k \neq l} c_{kl} s_l \frac{\partial G}{\partial s_k} + \sum_{k=1}^{n} c_O^k u_k \frac{\partial G}{\partial s_k} - G\left(\sum_{k=1}^{n} a_I^k\right) - \sum_{k \neq l} c_{kl} s_k \frac{\partial G}{\partial s_k} - \sum_{k=1}^{n} c_O^k s_k \frac{\partial G}{\partial s_k} \\
&= \sum_{k=1}^{n} \left( \sum_{l \neq k} c_{kl}\left(s_l - s_k\right) + c_O^k\left(u_k - s_k\right) \right) \frac{\partial G}{\partial s_k} + G \sum_{k=1}^{n} a_I^k \left(s_k - 1\right) \\
&= \left(\frac{\partial G}{\partial \boldsymbol{s}}\right)^T \left(-\boldsymbol{A}\left(\boldsymbol{s} - \mathbf{1}\right) + \operatorname{diag}\left(\boldsymbol{c}_O\right)\left(\boldsymbol{u} - \mathbf{1}\right)\right) + G \boldsymbol{a}_I^T \left(\boldsymbol{s} - \mathbf{1}\right), \qquad \text{(A.1.5)}
\end{aligned}$$

where

$$\boldsymbol{A} = \begin{pmatrix}
\sum\limits_{j \neq 1} c_{1j} + c_O^1 & -c_{12} & \cdots & -c_{1n} \\
-c_{21} & \sum\limits_{j \neq 2} c_{2j} + c_O^2 & \cdots & -c_{2n} \\
 & \vdots & & \\
-c_{n1} & -c_{n2} & \cdots & \sum\limits_{j \neq n} c_{nj} + c_O^n
\end{pmatrix}$$

and

$$\begin{aligned}
&\left(\boldsymbol{s} - \mathbf{1}\right)^T = \left(s_1 - 1, \ldots, s_n - 1\right), \quad \left(\boldsymbol{u} - \mathbf{1}\right)^T = \left(u_1 - 1, \ldots, u_n - 1\right) \\
&\left(\frac{\partial G}{\partial \boldsymbol{s}}\right)^T = \left(\frac{\partial G}{\partial s_1}, \ldots, \frac{\partial G}{\partial s_n}\right), \quad \boldsymbol{a}_I^T = \left(a_I^1, \ldots, a_I^n\right), \quad \boldsymbol{c}_O^T = \left(c_O^1, \ldots, c_O^n\right).
\end{aligned}$$

Here, $\operatorname{diag}\left(\boldsymbol{c}_O\right)$ is the diagonal matrix with diagonal elements $\left(c_O^1, \ldots, c_O^n\right)$.

This is a PDE for $G\left(s_1, \ldots, s_n, u_1, \ldots, u_n, t\right)$. To determine the solution, we also need an initial condition. Let us begin with the simple case that the system is

initially empty, i.e. all of the molecules come from the input channels $R_I^1, \ldots, R_I^n$. Thus at $t = 0$ we have $x_1 = \cdots = x_n = r_O^1 = \cdots = r_O^n = 0$. The initial condition is given by

$$G(s_1, \ldots, s_n, u_1, \ldots, u_n, 0) = \sum_{i_1,\ldots,i_n,j_1,\ldots,j_n} s_1^{i_1} \ldots s_n^{i_n} u_1^{j_1} \ldots u_n^{j_n} p_{i_1,\ldots,i_n,j_1,\ldots,j_n}(0)$$

$$= s_1^0 \ldots s_n^0 u_1^0 \ldots u_n^0 \times 1 = 1. \tag{A.1.6}$$

The solution for (A.1.5), (A.1.6) can be written as

$$G = e^{\boldsymbol{\lambda}^T(\boldsymbol{s}-\mathbf{1})+\boldsymbol{\lambda}_O^T(\boldsymbol{u}-\mathbf{1})} = \prod_{k=1}^{n} e^{\lambda_k(s_k-1)} \prod_{k=1}^{n} e^{\lambda_{Ok}(u_k-1)}, \tag{A.1.7}$$

where

$$\boldsymbol{\lambda}^T \triangleq (\lambda_1, \ldots, \lambda_n) = \boldsymbol{a}_I^T \left( \int_0^t e^{\boldsymbol{A}x} dx \right) e^{-\boldsymbol{A}t} \tag{A.1.8}$$

$$\boldsymbol{\lambda}_O^T \triangleq (\lambda_{O1}, \ldots, \lambda_{On}) = \boldsymbol{a}_I^T \left( \int_0^t e^{\boldsymbol{A}x} \int_x^t e^{-\boldsymbol{A}y} dy dx \right) \mathrm{diag}(\boldsymbol{c}_O). \tag{A.1.9}$$

In particular, if $\boldsymbol{A}$ is invertible and has $n$ linearly independent eigenvectors $\boldsymbol{v}_1^A, \ldots, \boldsymbol{v}_n^A$, with the corresponding eigenvalues $\lambda_1^A, \ldots, \lambda_n^A$, then (A.1.8) and (A.1.9) can be replaced by

$$\boldsymbol{\lambda}^T = \boldsymbol{a}_I^T \boldsymbol{V}_A \mathrm{diag} \left( \frac{1 - e^{-\lambda_i^A t}}{\lambda_i^A} \right) \boldsymbol{V}_A^{-1} \tag{A.1.10}$$

$$\boldsymbol{\lambda}_O^T = \left( \boldsymbol{a}_I^T t - \boldsymbol{\lambda}^T \right) \boldsymbol{A}^{-1} \mathrm{diag}(\boldsymbol{c}_O), \tag{A.1.11}$$

where $\boldsymbol{V}_A = \left( \boldsymbol{v}_1^A, \ldots, \boldsymbol{v}_n^A \right)$ is the matrix composed of the eigenvectors of $A$. $\mathrm{diag}(x_i) \triangleq \mathrm{diag}(\boldsymbol{x})$ where $\boldsymbol{x}^T = (x_1, \ldots, x_n)$.

We can easily obtain the generating function of $x_i, i = 1, \ldots, n$ and $r_O^i, i = 1, \ldots, n$ from their joint generating function (A.1.7) using (A.1.3):

$$G_{x_i} = G(1, \ldots, s_i, \ldots, 1) = e^{\lambda_i(s_i - 1)}, \quad G_{r_O^i} = G(1, \ldots, u_i, \ldots, 1) = e^{\lambda_{Oi}(u_i - 1)}.$$

Comparing with (A.1.1), we can see that $x_i$ is a Poisson random variable with parameter $\lambda_i$, and $r_O^i$ is a Poisson random variable with parameter $\lambda_{Oi}$. According to Theorem 1, (A.1.7) implies that $x_1, \ldots, x_n$, $r_O^1, \ldots, r_O^n$ are independent Poisson random variables

$$x_i \sim \mathcal{P}(\lambda_i), \; r_O^i \sim \mathcal{P}(\lambda_{Oi}), \; i = 1, \ldots, n. \tag{A.1.12}$$

The next problem is to find a way to sample those random variables in the simulation. The inputs $r_I^1, \ldots, r_I^n$ are just independent Poisson random variables with parameters $a_I^1 t, \ldots, a_I^n t$, so they are easy to sample. However when the inputs are sampled, we should not sample $x^i$ and $r_O^i$ directly from $\mathcal{P}(\lambda_i)$ and $\mathcal{P}(\lambda_{Oi})$. For example if we accidentally sampled a very large value for $x_i$ that it is even greater than the sum of all the inputs we sampled, then the result does not make sense. Instead we need to sample $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{r}_O = \left(r_O^1, \ldots, r_O^n\right)$ conditioned on $\boldsymbol{r}_I = \left(r_I^1, \ldots, r_I^n\right)$. In other words, we need to sample $\boldsymbol{x}$ and $\boldsymbol{r}_O$ using their conditional distribution when $\boldsymbol{r}_I$ is given.

Since the molecules coming from an input channel $R_I^i$ behave independently from molecules coming from other input channels, we can first focus on the molecules from $R_I^i$ and switch off $R_I^j, \; j \neq i$. Now we have only one input channel, and (A.1.8), (A.1.9) become (we have added the index $i$ to the notation to indicate that the values

are contributed by input channel $R_I^i$)

$$\left(\boldsymbol{\lambda}^i\right)^T = \left(\lambda_1^i, \ldots, \lambda_n^i\right) = a_I^i \boldsymbol{e}_i^T \left(\int_0^t e^{\boldsymbol{A}x} dx\right) e^{-\boldsymbol{A}t} \tag{A.1.13}$$

$$\left(\boldsymbol{\lambda}_O^i\right)^T = \left(\lambda_{O1}^i, \ldots, \lambda_{On}^i\right) = a_I^i \boldsymbol{e}_i^T \left(\int_0^t e^{\boldsymbol{A}x} \int_x^t e^{-\boldsymbol{A}y} dy dx\right) \mathrm{diag}\left(\boldsymbol{c}_O\right), \tag{A.1.14}$$

where $\boldsymbol{e}_i^T$ is the unit vector with the $i$th element being 1.

Now our purpose is to find the distributions of $\boldsymbol{x}$ and $\boldsymbol{r}_O$ when $r_I^i$ is given. The following theorem answers this question directly.

Theorem 2: If $X_i \sim \mathcal{P}\left(\lambda_i\right) (i = 1, \ldots, n)$ are independent Poisson random variables, then

$$X_i \left| \sum_{j=1}^n X_j \sim \mathcal{B}\left(\sum_{j=1}^n X_j,\ \frac{\lambda_i}{\sum_{j=1}^n \lambda_j}\right). \right.$$

*Proof.* We show the proof for $n = 2$. For $n > 2$, the problem can be converted to the $n = 2$ case using the fact that the sum of independent Poisson random variables is still a Poisson random variable.

As $X_1$ and $X_2$ are independent Poisson random variables

$$X_1 + X_2 \sim \mathcal{P}\left(\lambda_1 + \lambda_2\right) \Rightarrow P\left(X_1 + X_2 = n\right) = \frac{\left(\lambda_1 + \lambda_2\right)^n}{n!} e^{-(\lambda_1 + \lambda_2)}$$

$$P\left(X_1 = i \,|X_1 + X_2 = n\right) = \frac{P\left(X_1 = i\right) P\left(X_2 = n - i\right)}{P\left(X_1 + X_2 = n\right)}$$

$$= \frac{\lambda_1^i}{i!} e^{-\lambda_1} \frac{\lambda_2^{n-i}}{(n-i)!} e^{-\lambda_2} \Big/ \left(\frac{\left(\lambda_1 + \lambda_2\right)^n}{n!} e^{-(\lambda_1 + \lambda_2)}\right) = \frac{n!}{i!\,(n-i)!} \left(\frac{\lambda_1}{\lambda_1 + \lambda_2}\right)^i \left(\frac{\lambda_2}{\lambda_1 + \lambda_2}\right)^{n-i}$$

$$= P\left(Y = i\right),$$

where

$$Y \sim \mathcal{B}\left(n,\ \frac{\lambda_1}{\lambda_1 + \lambda_2}\right). \qquad \square$$

According to this theorem, the conditional distribution of $\boldsymbol{x}\left|r_I^i\right.$ and $\boldsymbol{r}_O\left|r_I^i\right.$ is actually a multinomial distribution:

$$\left(x_1,\ldots,x_n,r_O^1,\ldots,r_O^n\right)\left|r_I^i\right.\sim\mathcal{M}\left(r_I^i,\frac{\lambda_1^i}{a_I^it},\ldots,\frac{\lambda_n^i}{a_I^it},\frac{\lambda_{O1}^i}{a_I^it},\ldots,\frac{\lambda_{On}^i}{a_I^it}\right).$$

Here

$$a_I^it=\sum_{i=1}^n\lambda_i^i+\sum_{i=1}^n\lambda_{Oi}^i$$

because the sum of all of the Poisson random variables should be equal to the total input. This can also be verified in the following way. From (A.1.8) and (A.1.9), we have

$$\begin{aligned}\frac{d}{dt}\left(\boldsymbol{\lambda}^T\mathbf{1}+\boldsymbol{\lambda}_O^T\mathbf{1}\right)&=\boldsymbol{a}_I^T\left(\boldsymbol{I}-\left(\int_0^t e^{\boldsymbol{A}x}dx\right)e^{-\boldsymbol{A}t}\boldsymbol{A}\right)\mathbf{1}+\boldsymbol{a}_I^T\left(\int_0^t e^{\boldsymbol{A}x}e^{-\boldsymbol{A}t}dx\right)\operatorname{diag}\left(\boldsymbol{c}_O\right)\mathbf{1}\\&=\boldsymbol{a}_I^T\left(\mathbf{1}-\left(\int_0^t e^{\boldsymbol{A}x}dx\right)e^{-\boldsymbol{A}t}\boldsymbol{c}_O\right)+\boldsymbol{a}_I^T\left(\int_0^t e^{\boldsymbol{A}x}dx\right)e^{-\boldsymbol{A}t}\boldsymbol{c}_O=\boldsymbol{a}_I^T\mathbf{1}.\end{aligned}$$

Together with the initial condition $\boldsymbol{\lambda}(0)=\mathbf{0},\ \boldsymbol{\lambda}_O(0)=\mathbf{0}$, this yields

$$\boldsymbol{\lambda}^T\mathbf{1}+\boldsymbol{\lambda}_O^T\mathbf{1}=\left(\boldsymbol{a}_I^T\mathbf{1}\right)t$$

Now we can extend the result by switching on the other input channels. Since the molecules from different input channels do not interrupt each other, the result in this situation should be the sum all the multinomial random variables produced by each input channel,

$$\left(x_1,\ldots,x_n,r_O^1,\ldots,r_O^n\right)\left|\boldsymbol{r}_I\right.\sim\sum_{i=1}^n\mathcal{M}\left(r_I^i,\frac{\lambda_1^i}{a_I^it},\ldots,\frac{\lambda_n^i}{a_I^it},\frac{\lambda_{O1}^i}{a_I^it},\ldots,\frac{\lambda_{On}^i}{a_I^it}\right).\qquad\text{(A.1.15)}$$

Now let us remove the assumption that the system is initially empty. We also start from a simple case, assuming at time $t=0$ that we have $x_i\left(0\right)\neq0$, $x_j\left(0\right)=0\ \ (j\neq i)$.

Since these molecules have nothing to do with those coming from the input channels, we can switch off all the input channels and just look at the behavior of these molecules. Consider one such molecule. At any time $t > 0$, there is a probability $p_j^i(t)$ that the molecule stays at the state $S_j$. There is also a probability $p_{Oj}^i(t)$ that the molecule has already left the system through channel $R_O^j$. More importantly, these probabilities should be the same for every molecule that initially stays in $S_i$. Thus $(x_1(t), \ldots, x_n(t), r_1(t), \ldots, r_n(t))$ should have a multinomial distribution. To determine the parameters for this distribution, we need to compute $\boldsymbol{p}^i(t) \stackrel{\triangle}{=} \left(p_1^i(t), \ldots, p_n^i(t)\right)$ and $\boldsymbol{p}_O^i(t) \stackrel{\triangle}{=} \left(p_{O1}^i(t), \ldots, p_{On}^i(t)\right)$.

The master equation for a single molecule is given by

$$\frac{dp_j^i(t)}{dt} = \sum_{k \neq j} p_k^i(t) c_{kj} - p_j^i(t) \left( \sum_{k \neq j} c_{jk} + c_{Oj} \right) \tag{A.1.16}$$

$$\frac{dp_{Oj}^i(t)}{dt} = p_j^i(t) c_{Oj}, \quad j = 1, \ldots, n, \tag{A.1.17}$$

with initial condition

$$p_i^i(0) = 1, \quad p_j^i(0) = 0, \quad j \neq i \tag{A.1.18}$$

$$p_{Oj}^i(0) = 0, \quad j = 1 \ldots, n. \tag{A.1.19}$$

The solution to (A.1.16) and (A.1.18) is given by

$$\boldsymbol{p}^i(t) = e^{\boldsymbol{B}t} \boldsymbol{e}_i, \tag{A.1.20}$$

and from (A.1.17) and (A.1.19) we have

$$\boldsymbol{p}_O^i(t) = \int_0^t \operatorname{diag}(\boldsymbol{c}_O) e^{\boldsymbol{B}x} \boldsymbol{e}_i dx \tag{A.1.21}$$

where

$$\boldsymbol{B} = -\boldsymbol{A}^T.$$

If $\boldsymbol{B}$ has $n$ linearly independent eigenvectors $\boldsymbol{v}_1^B, \ldots, \boldsymbol{v}_n^B$, with the corresponding eigenvalues $\lambda_1^B, \ldots, \lambda_n^B$, then (A.1.20) and (A.1.21) can be replaced by

$$\boldsymbol{p}^i(t) = \boldsymbol{V}_B \mathrm{diag}\left(e^{\lambda_j^B t}\right) \boldsymbol{V}_B^{-1} \boldsymbol{e}_i \tag{A.1.22}$$

$$\boldsymbol{p}_O^i(t) = \mathrm{diag}(\boldsymbol{c}_O) \boldsymbol{V}_B \mathrm{diag}\left(\frac{e^{\lambda_j^B t} - 1}{\lambda_j^B}\right) \boldsymbol{V}_B^{-1} \boldsymbol{e}_i, \tag{A.1.23}$$

where $\boldsymbol{V}_B = \left(\boldsymbol{v}_1^B, \ldots, \boldsymbol{v}_n^B\right)$ is the matrix composed of the independent eigenvectors of $B$.

Putting all the molecules together, the distribution of $\boldsymbol{x}(t)$ and $\boldsymbol{r}_O(t)$ should be a multinomial distribution

$$(\boldsymbol{x}(t),\ \boldsymbol{r}_O(t)) \sim \mathcal{M}\left(x_i(0), \boldsymbol{p}^i(t), \boldsymbol{p}_O^i(t)\right).$$

Now we can let every species have a nonzero initial population. Since they do not influence each other, the result in this case should be the sum of all the multinomial random variables

$$(\boldsymbol{x}(t), \boldsymbol{r}_O(t)) \sim \sum_{i=1}^{n} \mathcal{M}\left(x_i(0), \boldsymbol{p}^i(t), \boldsymbol{p}_O^i(t)\right). \tag{A.1.24}$$

Having obtained the solution for the initial molecules, it is time to put everything together by switching on the input channels. The result in this case is the sum of

(A.1.15) and (A.1.24)

$$\left(\boldsymbol{x}\left(t\right),\boldsymbol{r}_{O}\left(t\right)\right)\sim\sum_{i=1}^{n}\mathcal{M}\left(x_{i}\left(0\right),\ \boldsymbol{p}^{i}\left(t\right),\ \boldsymbol{p}_{O}^{i}\left(t\right)\right)+\sum_{i=1}^{n}\mathcal{M}\left(r_{I}^{i},\ \frac{1}{a_{I}^{i}t}\boldsymbol{\lambda}^{i},\ \frac{1}{a_{I}^{i}t}\boldsymbol{\lambda}_{O}^{i}\right). \tag{A.1.25}$$

This is the time dependent solution for $\boldsymbol{x}\left(t\right)$ and $\boldsymbol{r}_{O}\left(t\right)$

For the simulations in Section III in the main paper, the mean and variance of $\boldsymbol{x}\left(t\right)$ have also been used. It would be convenient to have formulas for these values. It seems that we can compute them from (A.1.25), however (A.1.25) is the formula when $\boldsymbol{r}_{I}$ has already been sampled. If we need the mean and variance before $\boldsymbol{r}_{I}$ has been sampled, we must replace (A.1.15) by the Poisson random variables (A.1.12), yielding

$$\mathbb{E}\left(x_{i}\left(t\right)\right)=\sum_{j=1}^{n}x_{j}\left(0\right)p_{i}^{j}\left(t\right)+\lambda_{i} \tag{A.1.26}$$

$$\mathrm{Var}\left(x_{i}\left(t\right)\right)=\sum_{j=1}^{n}x_{j}\left(0\right)p_{i}^{j}\left(t\right)\left(1-p_{i}^{j}\left(t\right)\right)+\lambda_{i}. \tag{A.1.27}$$

In section III we also need to use the solutions for $n=1$ and $n=2$. The solutions for these two cases are given below.

$n=1$: In this case, $A=-B=c_{O}$, and $\lambda^{A}=-\lambda^{B}=c_{O}$. Equations (A.1.10) and (A.1.11) give

$$\lambda=\frac{a_{I}}{c_{O}}\left(1-e^{-c_{O}t}\right),\ \lambda_{O}=a_{I}t-\lambda,$$

and (A.1.22) and (A.1.23) yield

$$p\left(t\right)=e^{-c_{O}t},\ p_{O}\left(t\right)=1-e^{-c_{O}t}.$$

Thus the time dependent solution of $x\,(t)$ and $r_O\,(t)$ given by (A.1.25) is

$$(x\,(t)\,,\ r_O\,(t)) \sim \mathcal{M}\left(x\,(0)\,,\ e^{-c_O t},\ 1-e^{-c_O t}\right) + \mathcal{M}\left(r_I,\ \frac{1-e^{-c_O t}}{c_O t},\ 1-\frac{1-e^{-c_O t}}{c_O t}\right).$$

$n = 2$: Assume the two species are $E$ (enzyme) and $ES$ (enzyme-substrate compound) as shown in Figure 2.3. The population of $S$ (substrate) is very large ($x_S(0) \gg x_E(0),\ x_{ES}(0)$). The reactions in the system are

$$R_1:\ E+S \xrightarrow{c_1} ES,\quad R_2:\ ES \xrightarrow{c_2} E+S,\quad R_3:\ ES \xrightarrow{c_3} E+P$$

$$R_4:\ \phi \xrightarrow{a_4} E,\quad R_5:\ E \xrightarrow{c_5} \phi,\quad R_6:\ ES \xrightarrow{c_6} \phi.$$

During a stepsize of $S$, equation (A.1.25) in this case has the form

$$\begin{aligned}&\left(x_E(t),\ x_{ES}(t),\ r_O^E(t),\ r_O^{ES}(t)\right) \sim \mathcal{M}\left(x_E\,(0)\,,\ p_1^E(t),\ p_2^E(t),\ p_{O1}^E(t),\ p_{O2}^E(t)\right)\\ &+\mathcal{M}\left(x_{ES}\,(0)\,,\ p_1^{ES}(t),\ p_2^{ES}(t),\ p_{O1}^{ES}(t),\ p_{O2}^{ES}(t)\right) + \mathcal{M}\left(r_I^E, \frac{\lambda_1(t)}{a_I^E t},\ \frac{\lambda_2(t)}{a_I^E t},\ \frac{\lambda_{O1}(t)}{a_I^E t},\ \frac{\lambda_{O2}(t)}{a_I^E t}\right),\end{aligned} \tag{A.1.28}$$

where

$$(\lambda_1 \quad \lambda_2) = \left(a_I^E \quad a_I^{ES}\right)\left(\boldsymbol{v}_+^A \quad \boldsymbol{v}_-^A\right)\mathrm{diag}\left(\frac{1-e^{-\lambda_+^A t}}{\lambda_+^A},\ \frac{1-e^{-\lambda_-^A t}}{\lambda_-^A}\right)\left(\boldsymbol{v}_+^A \quad \boldsymbol{v}_-^A\right)^{-1}$$

$$(\lambda_{O1} \quad \lambda_{O2}) = \left(\left(a_I^E \quad a_I^{ES}\right)t - (\lambda_1 \quad \lambda_2)\right)\boldsymbol{A}^{-1}\mathrm{diag}\left(c_O^E, \quad c_O^{ES}\right)$$

$$\begin{pmatrix} p_1^E \\ \\ p_2^E \end{pmatrix} = \left(\boldsymbol{v}_+^B \;\; \boldsymbol{v}_-^B\right) \mathrm{diag}\left(e^{\lambda_+^B t},\; e^{\lambda_-^B t}\right) \left(\boldsymbol{v}_+^B \;\; \boldsymbol{v}_-^B\right)^{-1} \begin{pmatrix} 1 \\ \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} p_{O1}^E \\ \\ p_{O2}^E \end{pmatrix} = \mathrm{diag}\left(c_O^E,\; c_O^{ES}\right) \left(\boldsymbol{v}_+^B \;\; \boldsymbol{v}_-^B\right) \mathrm{diag}\left(\frac{e^{\lambda_+^B t}-1}{\lambda_+^B},\; \frac{e^{\lambda_-^B t}-1}{\lambda_-^B}\right) \left(\boldsymbol{v}_+^B \;\; \boldsymbol{v}_-^B\right)^{-1} \begin{pmatrix} 1 \\ \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} p_1^{ES} \\ \\ p_2^{ES} \end{pmatrix} = \left(\boldsymbol{v}_+^B \;\; \boldsymbol{v}_-^B\right) \mathrm{diag}\left(e^{\lambda_+^B t},\; e^{\lambda_-^B t}\right) \left(\boldsymbol{v}_+^B \;\; \boldsymbol{v}_-^B\right)^{-1} \begin{pmatrix} 0 \\ \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} p_{O1}^{ES} \\ \\ p_{O2}^{ES} \end{pmatrix} = \mathrm{diag}\left(c_O^E,\; c_O^{ES}\right) \left(\boldsymbol{v}_+^B \;\; \boldsymbol{v}_-^B\right) \mathrm{diag}\left(\frac{e^{\lambda_+^B t}-1}{\lambda_+^B},\; \frac{e^{\lambda_-^B t}-1}{\lambda_-^B}\right) \left(\boldsymbol{v}_+^B \;\; \boldsymbol{v}_-^B\right)^{-1} \begin{pmatrix} 0 \\ \\ 1 \end{pmatrix}.$$

Here,

$$a_I^E = a_4, \quad a_I^{ES} = 0, \quad c_O^E = c_5, \quad c_O^{ES} = c_6, \quad c_{E,ES} = c_1 x_S(0), \quad c_{ES,E} = c_2 + c_3$$

$$\boldsymbol{A} = -\boldsymbol{B}^T = \begin{pmatrix} c_{E,ES} + c_O^E & -c_{E,ES} \\ \\ -c_{ES,E} & c_{ES,S} + c_O^{ES} \end{pmatrix}, \tag{A.1.29}$$

where $\lambda_+^A$, $\lambda_-^A$, $\boldsymbol{v}_+^A$, $\boldsymbol{v}_-^A$ are the eigenvalues and corresponding eigenvectors of $\boldsymbol{A}$, and $\lambda_+^B$, $\lambda_-^B$, $\boldsymbol{v}_+^B$, $\boldsymbol{v}_-^B$ are the eigenvalues and corresponding eigenvectors of $\boldsymbol{B}$.

## A.2 The mean and variance of $Y = \mathcal{P}(X)$

Suppose that we sample two random variables $X$ and $Y$. $Y$ depends on $X$ in such a way that after we have sampled the value $x$ of $X$, we will sample $Y$ as a Poisson random variable with parameter $x$, i.e. $Y = \mathcal{P}(x)$. The purpose of this section is to compute the mean and variance of $Y$, and show that if we approximate $Y$ by $\mathcal{P}(\mathbb{E}X)$,

the approximation will give us the correct mean value but a smaller variance than the true $\mathrm{Var}\,(Y)$.

Let us begin with the expectation of $Y$. Using the conditional expectation, we have

$$\mathbb{E}Y = \mathbb{E}\left(\mathbb{E}\left(Y\,|X\,\right)\right).$$

When $X$ is given, $Y$ is a Poisson random number with parameter $X$, so the conditional expectation $\mathbb{E}\left(Y\,|X\,\right)$ is actually the expectation of a Poisson random variable with the given parameter $X$. Thus,

$$\mathbb{E}\left(Y\,|X\,\right) = X$$

and

$$\mathbb{E}Y = \mathbb{E}\left(\mathbb{E}\left(Y\,|X\,\right)\right) = \mathbb{E}X. \tag{A.2.1}$$

For the variance of $Y$, we have

$$\mathrm{Var}\,(Y) = \mathbb{E}\left(Y^2\right) - \left(\mathbb{E}Y\right)^2. \tag{A.2.2}$$

For $\mathbb{E}\left(Y^2\right)$ we also use the conditional expectation

$$\mathbb{E}\left(Y^2\right) = \mathbb{E}\left(\mathbb{E}\left(Y^2\,|X\,\right)\right). \tag{A.2.3}$$

Here

$$\mathbb{E}\left(Y^2\,|X\,\right) = \mathrm{Var}\left(Y\,|X\,\right) + \left(\mathbb{E}\left(Y\,|X\,\right)\right)^2 = X + X^2. \tag{A.2.4}$$

The last step in the previous equation uses the fact that when $X$ is given, $Y$ is a Poisson random variable with parameter $X$ so both the mean and the variance of $Y$ are equal

to $X$. Inserting (A.2.4) in (A.2.3) yields

$$\mathbb{E}\left(Y^2\right) = \mathbb{E}\left(\mathbb{E}\left(Y^2\,|X\right)\right) = \mathbb{E}\left(X + X^2\right) = \mathbb{E}X + \mathbb{E}\left(X^2\right).$$

Inserting this into (A.2.2) and using (A.2.1), we obtain the variance of $Y$,

$$\mathrm{Var}\left(Y\right) = \mathbb{E}X + \mathbb{E}\left(X^2\right) - \left(\mathbb{E}Y\right)^2 = \mathbb{E}X + \mathbb{E}\left(X^2\right) - \left(\mathbb{E}X\right)^2 = \mathbb{E}X + \mathrm{Var}\left(X\right). \quad \text{(A.2.5)}$$

Now we can compare this with the approximation $Y' = \mathcal{P}\left(\mathbb{E}X\right)$. As $\mathbb{E}X$ is a real number, $Y'$ is actually a Poisson random variable with

$$\mathbb{E}\left(Y'\right) = \mathrm{Var}\left(Y'\right) = \mathbb{E}X.$$

Comparing this with (A.2.1) and (A.2.5), we can see that the approximation has the same mean value but a smaller variance.

## A.3 The mean and variance of the number of firings in a reaction channel

Consider the Example System from Appendix A.1. For any species in $\hat{S}$, we know its time dependent solution. Thus there is no stepsize requirement associated with this species, as long as the species not belonging to $\hat{S}$ can be considered as constant. In another words, we need only to compute the stepsize for species not in $\hat{S}$.

We use the following inequalities to bound the change of a species.

$$\mathbb{E}\Delta x_i \leq \max\left(\frac{\epsilon}{g_i}x_i, 1\right), \quad \sqrt{\mathrm{Var}\left(\Delta x_i\right)} \leq \max\left(\frac{\epsilon}{g_i}x_i, 1\right),$$

where $g_i$ is a constant that depends on the highest order of the reactions which involve $S_i$ as a reactant. In the current situation $r_i$ may no longer be a Poisson random variable. The purpose of this section is to find the mean and variance for such reactions.

For the system in Appendix A.1, we can partition the reactions into three groups:

1. Reactions whose reactants do not belong to $\hat{S}$ (e.g. all the input channels). As the reactants for these reactions can be considered constant during the step, these reactions can be sampled by Poisson random variables as in tau leaping.

2. Reactions corresponding to output channels. In the Example System of Appendix A.1, the output reactions are $R_O^i,\ i = 1, \ldots, n$, however, generally speaking a species $S_i \in \hat{S}$ could have several output reactions, i.e. $R_O^i$ in not just one reaction but a set of reactions. These reactions should compete with each other for a share of $r_O^i$. Now the rate constant $c_O^i$ for $R_O^i$ is the sum of all the rate constants for reactions in $R_O^i$. Supposing that $R_k : S_i \to \phi$ is in $R_O^i$ with reaction rate $c_k$. Then the probability that $R_k$ is responsible for a firing of $R_O^i$ is $c_k/c_O^i$.

   Now let us compute the mean and variance of $r_k$. In the Example System of Appendix A.1, there are $r_O^i$ molecules consumed by $R_O^i$. These molecules come from either the input channels (denoted by $r_P^i$) or the initial molecules of species in $\hat{S}$ (denoted by $r_B^i$). Thus

$$r_O^i = r_P^i + r_B^i.$$

It is shown in Appendix A.1 that $r_P^i$ is a Poisson random number with parameter $\lambda_{Oi}$ (see (A.1.12)),

$$r_P^i \sim \mathcal{P}\left(\lambda_{Oi}\right).$$

$r_B^i$ is the sum of $n$ binomial random variables with parameters $\left(x_j\left(0\right),\ p_{Oi}^j\right)$, $j = 1, \ldots, n.$ (see (A.1.24)),

$$r_B^i \sim \sum_{j=1}^{n} \mathcal{B}\left(x_j\left(0\right),\ p_{Oi}^j\right).$$

We want to distribute these molecules to the output channels in $R_O^i$. The probability that a molecule goes through reaction channel $R_k$ is $c_k/c_O^i$. To distribute the first part, we make use of the following theorem.

Theorem 3. Let $N$ be a Poisson random number with parameter $\lambda$. Then the sum of $N$ i.i.d Bernoulli variables with parameter $p$ is also a Poisson random variable with parameter $\lambda p$.

The proof can be found in a probability textbook (see example (27) in [35]).

In our case, $r_P^i \sim \mathcal{P}\left(\lambda_{Oi}\right)$, and each molecule in $r_P^i$ has a probability $c_k/c_O^i$ to go through channel $R_k$. By Theorem 3, the number of molecules that choose $R_k$ is a Poisson random number

$$\mathcal{P}\left(\frac{c_k}{c_O^i}\lambda_{Oi}\right).$$

Now let us distribute the second part $r_B^i$. $r_B^i$ is the sum of $n$ independent binomial random numbers. Each molecule in $r_B^i$ also has a probability $c_k/c_O^i$ to choose channel $R_k$, so in this case the number of molecules $R_k$ consumed is also the sum

of $n$ binomial random variables

$$\sum_{j=1}^{n} \mathcal{B}\left(x_j\left(0\right),\ \frac{c_k}{c_O^i} p_{Oi}^j\right).$$

Adding the two parts together, we obtain

$$r_k \sim \mathcal{P}\left(\frac{c_k}{c_O^i}\lambda_{Oi}\right) + \sum_{j=1}^{n} \mathcal{B}\left(x_j\left(0\right),\ \frac{c_k}{c_O^i} p_{Oi}^j\right).$$

The mean and variance of $r_k$ can be calculated by

$$\mathbb{E}r_k = \frac{c_k}{c_O^i}\left(\lambda_{Oi} + \sum_{j=1}^{n} x_j\left(0\right) p_{Oi}^j\right), \mathrm{Var}\left(r_k\right) = \frac{c_k}{c_O^i}\left(\lambda_{Oi} + \sum_{j=1}^{n} x_j\left(0\right) p_{Oi}^j\left(1 - \frac{c_k}{c_O^i} p_{Oi}^j\right)\right).$$

3. Reactions which convert one species in $\hat{S}$ to another species in $\hat{S}$. In Appendix A.1, the $R_{ij},\ i,j = 1,\ldots,n$ are of this type. In a more general case, $R_{ij}$ can contain several reactions as well. Suppose that $R_k : S_i \to S_j$ is one of them, with rate constant $c_k$.

   Now we want to compute the mean and variance of $r_k$. Since we use species $S_i$ as the reactant and its population is a random variable during the step, we may not have an exact formula for $r_k$. Here we use the following approximation,

$$r_k \approx \mathcal{P}\left(c_k\left(\int_0^\tau \mathbb{E}x_i\left(t\right)dt + \frac{\tau}{2}\left(x_i\left(\tau\right) - \mathbb{E}\left(x_i\left(\tau\right)\right)\right)\right)\right). \tag{A.3.1}$$

   The mean and variance of $r_k$ can be computed using (A.2.1) and (A.2.5) as follows:

$$\mathbb{E}r_k \approx \mathbb{E}\left(c_k\left(\int_0^\tau \mathbb{E}x_i\left(t\right)dt + \frac{\tau}{2}\left(x_i\left(\tau\right) - \mathbb{E}\left(x_i\left(\tau\right)\right)\right)\right)\right) = c_k\int_0^\tau \mathbb{E}\left(x_k\left(t\right)\right)dt$$

$$\mathrm{Var}\left(r_k\right) \approx c_k\int_0^\tau \mathbb{E}\left(x_i\left(t\right)\right)dt + \mathrm{Var}\left(c_k\left(\int_0^\tau \mathbb{E}x_i\left(t\right)dt + \frac{\tau}{2}\left(x_i\left(\tau\right) - \mathbb{E}\left(x_i\left(\tau\right)\right)\right)\right)\right)$$

$$= c_k\int_0^\tau \mathbb{E}\left(x_i\left(t\right)\right)dt + \frac{\tau^2}{4}\mathrm{Var}\left(x_i\left(\tau\right)\right).$$

   Here the formulas for $\mathbb{E}\left(x_i\left(t\right)\right)$ and $\mathrm{Var}\left(x_i\left(t\right)\right)$ are given by (A.1.26) and (A.1.27).

## A.4 Sampling a feasible flow in the network

Consider each species in $\hat{S}$ as a vertex. Vertices $i$ and $j$ are connected if there are reactions which convert species $S_i$ to $S_j$ or $S_j$ to $S_i$. On each edge we define the flow

$$f_{ij} = r_{ij} - r_{ji}, \tag{A.4.1}$$

where $f_{ij}$ indicates the number of molecules that go from $S_i$ to $S_j$. If its value is negative, there are more firings of $R_{ji}$ than $R_{ij}$.

Using the result in Appendix A.1, we can sample all the input reactions $R_I^i$, all the output reactions $R_O^i$ and the population vector $\boldsymbol{x}$. However sometimes we also need to sample the reactions $R_{ij}$. If we do this, we should make sure that we only sample the flow for a proper set of edges. Here 'a proper set' means that after sampling the flow values for this set, the flow values of other edges can be uniquely determined by mass conservation equations.

For each vertex $i$, the mass conservation equation is given by,

$$r_I^i + x_i(0) = x_i(t) + r_O^i + \sum_{j \neq i} f_{ij}. \tag{A.4.2}$$

Consider a connected subgraph $G = (V,\ E)$, where $V$ is the set of vertices in $G$ and $E$ is the set of edges in $G$. Each vertex provides a mass conservation equation and each edge provides an unknown. If the subgraph contains no loop, then the number of vertices is one more than the number of edges, which means that the number of equations is one more than the number of unknowns. However, these equations are not independent.

Summing (A.4.2) up over all vertices in $V$, we obtain

$$\sum_{i \in V} \left( r_I^i + x_i \left( 0 \right) \right) = \sum_{i \in V} \left( x_i \left( t \right) + r_O^i \right).$$

Here the flows completely cancel out. This equation simply shows the total mass conservation of the system and it is automatically satisfied by (A.1.25). Thus the number of independent equations is one less than the total number of vertices in $V$. For the connected subgraph $G$ we have the same number of equations and unknowns, thus the flow can be determined.

After obtaining a flow value $f_{ij}$ from the mass conservation equation, we can go on sampling $r_{ij}$ and $r_{ij}$ in the following manner such that (A.4.1) is satisfied:

If $f_{ij} \geq 0$, sample $r_{ji}$ using (A.3.1) and compute $r_{ij}$ as $r_{ij} = r_{ji} + f_{ij}$. If $f_{ij} < 0$, sample $r_{ij}$ using (A.3.1) and compute $r_{ji}$ as $r_{ji} = r_{ij} - f_{ij}$.

If $G$ has loops, the number of unknowns will be more than the number of equations. In this case, we need to sample the flow value (by sampling $r_{ij}$ and $r_{ji}$ using (A.3.1) and computing $f_{ij}$ using (A.4.1)) of some edges to decrease the number of unknown. The following is a simple algorithm to determine the edges we are going to sample.

1. Create an empty list $L$. Arbitrarily pick a start vertex $i$ in $V$ and push it into $L$. Create a pointer and let it point to the first element in the list, which at the beginning is $i$.

2. Push all the vertices connected to $i$ into the list.

3. Move the pointer to the next element in the list (suppose the second element is $j$).

4. Collect all the vertices connected to $j$ except the one that caused $j$ to have been pushed into the list, i.e. the vertex $i$. Denote these vertices by $V_j$.

5. Compare every vertex in $V_j$ with the elements in the list. If a vertex $k \in V_j$ is not in the list, push it into the list. If it is already in the list, this implies that there is a loop in the system. This is because we already have a path from $i$ to $k$ and now we have found another one. It is obvious that edge $e_{jk}$ is in the loop, so we sample the value of $f_{jk}$ and cut the edge $e_{jk}$. Now we have removed the loop. Continue comparing other vertices until all the vertices in $V_j$ are treated as we do for vertex $k$.

6. Move the pointer to the next element in the list and do the same as we did for vertex $j$. Stop the process when the pointer has walked through the whole list.

After applying the above algorithm to the graph $G$, the unsampled edges contain no loops. Thus we have the same number of independent equations and unknowns, and the flow in the graph can be uniquely determined.

## A.5 Solution to the master equation for a one dimensional discrete diffusion process

In this section we derive the probability that a molecule jumps from one voxel to another in a 1D domain. Suppose we discretize a 1D domain into $L$ voxels with reflecting boundary conditions, and that there is a single molecule in the domain. The probability that the molecule jumps to a particular neighbor voxel in the next infinitesimal $dt$ is $\kappa dt$. Define $p_{i,j}(t)$ as the probability that the molecule jumps from voxel $i$ to voxel $j$ after a time interval $t$. Then $p_{i,j}(t)$ satisfies the following equation

$$\begin{aligned} p_{i,j}(t+dt) &= p_{i,j-1}(t)\,\kappa dt + p_{i,j+1}(t)\,\kappa dt + p_{i,j}(t)\,(1-2\kappa dt) \\ \Longrightarrow \frac{d}{dt}p_{i,j}(t) &= \kappa\left(p_{i,j-1}(t) + p_{i,j+1}(t) - 2p_{i,j}(t)\right), \quad j = 2,\ldots,L-1 \end{aligned}$$

For $j = 1$ or $j = L$ we have

$$\frac{d}{dt}p_{i,1}(t) = \kappa\left(p_{i,2}(t) - p_{i,1}\right), \quad \frac{d}{dt}p_{i,L}(t) = \kappa\left(p_{i,L-1}(t) - p_{i,L}\right).$$

Rewriting in a more compact form yields

$$\frac{d}{dt}\begin{pmatrix} p_{i,1}(t) \\ p_{i,2}(t) \\ \vdots \\ p_{i,L-1}(t) \\ p_{i,L}(t) \end{pmatrix} = \kappa \begin{pmatrix} -1 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \end{pmatrix} \begin{pmatrix} p_{i,1}(t) \\ p_{i,2}(t) \\ \vdots \\ p_{i,L-1}(t) \\ p_{i,L}(t) \end{pmatrix}, \tag{A.5.1}$$

with initial condition

$$p_{i,j}(0) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}. \tag{A.5.2}$$

The eigenvalues of the coefficient matrix in (A.5.1) are

$$\lambda_i = 2\left(\cos\frac{i\pi}{L} - 1\right), \quad i = 0, \ldots, L-1 \tag{A.5.3}$$

and the corresponding eigenvectors $\mathbf{v}_i = (v_{i1}, \ldots, v_{iL})^T$ have elements

$$v_{ij} = \cos\left(\frac{i\pi}{2L} - \frac{ij\pi}{L}\right), \quad i = 0, \ldots, L-1; \quad j = 1, \ldots, L. \tag{A.5.4}$$

The solution of the ODE (A.5.1) has the form

$$\begin{pmatrix} p_{i,1}(t) \\ \vdots \\ p_{i,L}(t) \end{pmatrix} = \mathbf{V} \begin{pmatrix} e^{\kappa\lambda_0 t} & & \\ & \ddots & \\ & & e^{\kappa\lambda_{L-1}t} \end{pmatrix} \mathbf{V}^{-1} \begin{pmatrix} p_{i,1}(0) \\ \vdots \\ p_{i,L}(0) \end{pmatrix}, \tag{A.5.5}$$

where

$$\mathbf{V} = (\mathbf{v}_0, \ldots, \mathbf{v}_{L-1}) \tag{A.5.6}$$

is the matrix consisting of the eigenvectors. In the simulation it is convenient to normalize the eigenvectors, in which case $V$ becomes a unit orthogonal matrix and the inverse operation in (A.5.5) can be replaced by a transpose operation.

Equation (A.5.5) gives the probability that a molecule jumps from voxel $i$ to voxel $j$ after a time interval $t$ with reflecting boundary conditions. For some other common boundary conditions, the jump probabilities can be expressed similarly. Consider,

for example, the case of periodic boundary conditions. Since the first and the last voxels are adjacent, the ODE system for $p_{ij}(t)$ becomes

$$\frac{d}{dt}\begin{pmatrix} p_{i,1}(t) \\ p_{i,2}(t) \\ \vdots \\ p_{i,L-1}(t) \\ p_{i,L}(t) \end{pmatrix} = \kappa \begin{pmatrix} -2 & 1 & & & 1 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 1 & & & 1 & -2 \end{pmatrix} \begin{pmatrix} p_{i,1}(t) \\ p_{i,2}(t) \\ \vdots \\ p_{i,L-1}(t) \\ p_{i,L}(t) \end{pmatrix}. \tag{A.5.7}$$

The eigenvalues of the coefficient matrix are

$$\lambda_i = 2\left(\cos\left(\frac{2i\pi}{L}\right) - 1\right), \quad i = 0, \ldots, \left\lfloor \frac{L}{2} \right\rfloor, \tag{A.5.8}$$

and the corresponding eigenvectors $\mathbf{u}_i = (u_{i1}, \ldots, u_{iL})^T$, $\mathbf{v}_i = (v_{i1}, \ldots, v_{iL})^T$ for $\lambda_i$ are

$$\begin{aligned} u_{ij} &= \sin\left(\frac{2ij\pi}{L}\right), \quad i = 1, \ldots, \left\lceil \frac{L}{2} \right\rceil - 1, \quad j = 1, \ldots, L, \\ v_{ij} &= \cos\left(\frac{2ij\pi}{L}\right), \quad i = 0, \ldots, \left\lfloor \frac{L}{2} \right\rfloor, \quad j = 1, \ldots, L. \end{aligned} \tag{A.5.9}$$

Thus the solution to (A.5.7) is given by

$$\begin{pmatrix} p_{i,1}(t) \\ \vdots \\ p_{i,L}(t) \end{pmatrix} = \mathbf{V} \begin{pmatrix} e^{\kappa\lambda_0 t} & & & & & \\ & e^{\kappa\lambda_1 t} & & & & \\ & & e^{\kappa\lambda_1 t} & & & \\ & & & e^{\kappa\lambda_2 t} & & \\ & & & & e^{\kappa\lambda_2 t} & \\ & & & & & \ddots \end{pmatrix} \mathbf{V}^{-1} \begin{pmatrix} p_{i,1}(0) \\ \vdots \\ p_{i,L}(0) \end{pmatrix}, \tag{A.5.10}$$

where

$$\mathbf{V} = (\mathbf{v}_0, \mathbf{u}_1, \mathbf{v}_1, \mathbf{u}_2, \mathbf{v}_2, \dots).$$

In Section 3.4.3 we need the solution of a diffusion process with absorbing boundary conditions. The ODE system is given by

$$\frac{d}{dt}\begin{pmatrix} p_{i,1}(t) \\ p_{i,2}(t) \\ \vdots \\ p_{i,L-1}(t) \\ p_{i,L}(t) \end{pmatrix} = \kappa \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix} \begin{pmatrix} p_{i,1}(t) \\ p_{i,2}(t) \\ \vdots \\ p_{i,L-1}(t) \\ p_{i,L}(t) \end{pmatrix}. \tag{A.5.11}$$

The eigenvalues of the coefficient matrix are

$$\lambda_i = 2\left(\cos\left(\frac{i\pi}{L+1}\right) - 1\right), \quad i = 1, \dots, L,$$

and the corresponding eigenvectors $\mathbf{v}_i = (v_{i1}, \dots, v_{iL})^T$ for $\lambda_i$ are

$$v_{ij} = \sin\left(\frac{ij\pi}{L+1}\right), \quad i = 1, \dots, L, \quad j = 1, \dots, L.$$

The solution to (A.5.11) is given by

$$\begin{pmatrix} p_{i,1}(t) \\ \vdots \\ p_{i,L}(t) \end{pmatrix} = \mathbf{V} \begin{pmatrix} e^{\kappa\lambda_1 t} & & \\ & \ddots & \\ & & e^{\kappa\lambda_L t} \end{pmatrix} \mathbf{V}^{-1} \begin{pmatrix} p_{i,1}(0) \\ \vdots \\ p_{i,L}(0) \end{pmatrix}, \tag{A.5.12}$$

where

$$\mathbf{V} = (\mathbf{v}_1, \dots, \mathbf{v}_L).$$

## A.6 Derivation of the upper bound of $\mathrm{E}\left(p(\tau)\right)$

Let us look at a particular molecule that is a reactant in one or more reactions in the system. We can divide all possible reaction events into two groups $\mathcal{R}$ and $\overline{\mathcal{R}}$, where $\mathcal{R}$ is the set of possible reaction events in which the observed molecule is involved as a reactant and $\overline{\mathcal{R}}$ the set of possible reaction events that the observed molecule is not involved. Denote by $p_r\left(t\right)$ the probability that a reaction event in $\mathcal{R}$ occurs before time $t$, given that no events in $\overline{\mathcal{R}}$ occur before $t$. We seek an upper bound on the expectation of $p_r\left(\tau\right)$, where $\tau$ is also a random variable which is defined as the time when the first reaction event of the system occurs. In another words, we seek an upper bound for $\mathrm{E}\left(p_r\left(\tau\right)\right)$.

Denote by $q_r\left(t\right) = 1 - p_r\left(t\right)$ the probability that no reaction event in $\mathcal{R}$ occurs before $t$, i.e. the observed molecule does not react before time $t$, given that no events in $\overline{\mathcal{R}}$ occur before $t$. To simplify the notation, we denote $(\mathcal{U}, [t_0, t_1])$ as the event that no reaction in $\mathcal{U}$ occurs during $[t_0, t_1]$ ($\mathcal{U}$ could be $\mathcal{R}$, $\overline{\mathcal{R}}$, etc), thus $q_r(t)$ is equivalent

to $P\left((\mathcal{R},[0,t])\,\middle|\,(\overline{\mathcal{R}},[0,t])\right)$, and we have

$$
\begin{aligned}
&q_r\left(t+dt\right)P\left((\overline{\mathcal{R}},[t,t+dt])\,\middle|\,(\overline{\mathcal{R}},[0,t])\right)P\left((\overline{\mathcal{R}},[0,t])\right)\\
=&P\left((\mathcal{R},[0,t+dt])\,\middle|\,(\overline{\mathcal{R}},[0,t+dt])\right)P\left((\overline{\mathcal{R}},[0,t+dt])\right)\\
=&P\left((\mathcal{R}\cup\overline{\mathcal{R}},[0,t+dt])\right)\\
=&P\left((\mathcal{R}\cup\overline{\mathcal{R}},[0,t])\right)P\left((\mathcal{R}\cup\overline{\mathcal{R}},[t,t+dt])\,\middle|\,(\mathcal{R}\cup\overline{\mathcal{R}},[0,t])\right)\\
=&P\left((\mathcal{R},[0,t])\,\middle|\,(\overline{\mathcal{R}},[0,t])\right)P\left((\overline{\mathcal{R}},[0,t])\right)P\left((\mathcal{R}\cup\overline{\mathcal{R}},[t,t+dt])\,\middle|\,(\mathcal{R}\cup\overline{\mathcal{R}},[0,t])\right)\\
=&q_r(t)P\left((\overline{\mathcal{R}},[0,t])\right)P\left((\mathcal{R}\cup\overline{\mathcal{R}},[t,t+dt])\,\middle|\,(\mathcal{R}\cup\overline{\mathcal{R}},[0,t])\right),
\end{aligned}
$$

which implies

$$
q_r\left(t+dt\right)=q_r(t)\frac{P\left((\mathcal{R}\cup\overline{\mathcal{R}},[t,t+dt])\,\middle|\,(\mathcal{R}\cup\overline{\mathcal{R}},[0,t])\right)}{P\left((\overline{\mathcal{R}},[t,t+dt])\,\middle|\,(\overline{\mathcal{R}},[0,t])\right)}. \tag{A.6.1}
$$

Here the numerator on the right hand side is the probability that no reaction event occurs during $[t,t+dt]$, given that no reaction occurs before $t$. Thus it equals $1-a_0(t)dt$ where $a_0(t)$ is the total propensity of the system at time $t$ given that no reaction occurs before $t$. Similarly, the denominator equals $1-a_{\bar{\mathcal{R}}}(t)dt$ where $a_{\bar{\mathcal{R}}}(t)$ is defined as the propensity of events in $\overline{\mathcal{R}}$ at time $t$ given that no events in $\overline{\mathcal{R}}$ occur before $t$. Thus (A.6.1) can be reduced to

$$
\begin{aligned}
&q_r\left(t+dt\right)=q_r(t)\frac{P\left((\mathcal{R}\cup\overline{\mathcal{R}},[t,t+dt])\,\middle|\,(\mathcal{R}\cup\overline{\mathcal{R}},[0,t])\right)}{P\left((\overline{\mathcal{R}},[t,t+dt])\,\middle|\,(\overline{\mathcal{R}},[0,t])\right)}\\
=&q_r(t)\frac{1-a_0(t)dt}{1-a_{\bar{\mathcal{R}}}(t)dt}=q_r(t)\left(1-a_0(t)dt\right)\left(1+a_{\bar{\mathcal{R}}}(t)dt+O(dt^2)\right)\\
=&q_r(t)\left(1-(a_0(t)-a_{\bar{\mathcal{R}}}(t))dt\right)+O(dt^2)\triangleq q_r(t)\left(1-a(t)dt\right)+O(dt^2).
\end{aligned}
\tag{A.6.2}
$$

Here $a(t) = a_0(t) - a_{\bar{\mathcal{R}}}(t)$ is the difference between the total propensity and the propensity of the reaction events involving only molecules other than the observed one. Thus it can be considered as the contribution that the observed molecule gives to the total propensity. (A.6.2) yields an ODE

$$\frac{d}{dt}q_r(t) = -q_r(t)a(t)$$

whose solution is

$$q_r(t) = e^{-\int_0^t a(s)ds}.$$

Thus the probability for the observed molecule to be involved in a reaction event before time $t$, under the condition that no other reaction event occurs before $t$, is given by

$$p_r(t) \triangleq 1 - q_r(t) = 1 - e^{-\int_0^t a(s)ds}. \tag{A.6.3}$$

To estimate $\mathrm{E}\left(p_r\left(\tau\right)\right)$, it is necessary to find the distribution of $\tau$. Define $q\left(t\right)$ to be the probability that the system does not fire a reaction before time $t$. As $a_0\left(t\right)dt$ is the probability that the system fires a reaction in the infinitesimal $[t, t+dt]$ given that no reaction occurs before $t$, then we obtain

$$q\left(t+dt\right) = q\left(t\right)\left(1 - a_0\left(t\right)dt\right) \implies q\left(t\right) = e^{-\int_0^t a_0(s)ds}$$

$$p\left(t\right) \triangleq 1 - q\left(t\right) = 1 - e^{-\int_0^t a_0(s)ds},$$

where $p\left(t\right)$ is the probability that the system fires the first reaction before $t$.

Now we can estimate $\mathrm{E}(p_r(\tau))$ as

$$\begin{aligned}
&\mathrm{E}(p_r(\tau)) = \int_0^\infty p_r(t)\, dp(t) = p_r(0) + \int_0^\infty p_r'(t)\,(1-p(t))\, dt \\
&= \int_0^\infty a(t)\, e^{-\int_0^t (a(s)+a_0(s))ds} dt = \int_0^\infty \frac{a(t)}{a(t)+a_0(t)}\,(a(t)+a_0(t))\, e^{-\int_0^t (a(s)+a_0(s))ds} dt \\
&\leq \max_{t>0} \frac{a(t)}{a(t)+a_0(t)} \int_0^\infty e^{-\int_0^t (a(s)+a_0(s))ds}\,(a(t)+a_0(t))\, dt \\
&= \max_{t>0} \frac{a(t)}{a(t)+a_0(t)}\, e^{-\int_0^t (a(s)+a_0(s))ds}\Big|_\infty^0 = \max_{t>0} \frac{a(t)}{a(t)+a_0(t)} = \frac{1}{1+\min_{t>0}\frac{a_0(t)}{a(t)}}.
\end{aligned} \tag{A.6.4}$$

Here the second equality uses integration by parts. Equation (A.6.4) shows that an upper bound of $\mathrm{E}(p_r(\tau))$ is determined by the ratio of the total propensity of the system and the propensity contributed by the observed molecule. $\mathrm{E}(p_r(\tau))$ will be a small value when the ratio is large. It is worth mentioning that this value cannot be controlled by decreasing the "stepsize". This is because the stepsize of this simulation is the time to the next chemical reaction event, which is determined by the behavior of the system rather than a value that can be manipulated at will.

## A.7 A discussion about the probability that a molecule diffuses to a given subvolume

Suppose we have two one dimensional systems, system 1 and system 2, with reflecting boundary conditions. The two systems are initially identical except that molecules in system 2 are inert, thus that system is simply governed by a diffusion process. Let us look at two molecules of the same species that initially are at the same

position but in the different systems. For the observed molecule in system 1, let $\mathcal{R}$ be the set of possible reaction events in which the observed molecule is involved as a reactant and $\overline{\mathcal{R}}$ be the set of possible reaction events in which the observed molecule is not involved. Suppose the two molecules are initially in voxel $i$. Define $\hat{p}_{i,j}(t)$ as the probability that the molecule in system 1 diffuses to voxel $j$ at time $t$, under the condition that no event in $\overline{\mathcal{R}}$ occurs before $t$, and $p_{i,j}(t)$ the probability of the same event for the molecule in system 2. The purpose of this section is to show that $\hat{p}_{i,j}(t) \leq p_{i,j}(t)$.

Intuitively it is obvious that $\hat{p}_{i,j}(t) \leq p_{i,j}(t)$ because in system 1 the fact that the molecule been observed in voxel $j$ at time $t$ means that it not only has diffused to voxel $j$ but also survived (from reaction) up to time $t$, thus the probability should be smaller that the value given by system 2 in which the molecules always survive. Here we provide a more rigorous proof of that fact.

Denote the probability that the molecule jumps to a particular neighbor voxel in a infinitesimal $dt$ by $\kappa dt$. Then in system 2, the diffusion process gives the equation (A.5.1) with initial condition (A.5.2). However for system 1 which includes reactions, $\hat{p}_{i,j}(t)$ satisfies

$$\begin{aligned}
&\hat{p}_{i,j}(t+dt) = \hat{p}_{i,j-1}(t)\,\kappa dt + \hat{p}_{i,j+1}(t)\,\kappa dt + \hat{p}_{i,j}(t)\,(1 - 2\kappa dt - a_j(t)\,dt) \\
&\implies \frac{d}{dt}\hat{p}_{i,j}(t) = \kappa\left(\hat{p}_{i,j-1}(t) + \hat{p}_{i,j+1}(t) - 2\hat{p}_{i,j}(t)\right) - a_j(t)\,\hat{p}_{i,j}(t),
\end{aligned}$$

where $a_j(t)$ is the propensity contributed by the molecule, which is defined as $a_j(t) = a_0(j,t) - a_{\bar{\mathcal{R}}}(t)$ where $a_0(j,t)$ is the total propensity of the system at time $t$ given that no reaction occurs before $t$ and the observed molecule is in voxel $j$ at time $t$, and $a_{\bar{R}}(t)$

is the total propensity of events in $\overline{\mathcal{R}}$ at time $t$ given that no event in $\overline{\mathcal{R}}$ occurs before $t$. Denoting $b_j(t) = a_j(t)\,\hat{p}_{i,j}(t)$, then $\hat{p}_{i,j}(t)$ satisfies

$$\frac{d}{dt}\begin{pmatrix} \hat{p}_{i,1}(t) \\ \hat{p}_{i,2}(t) \\ \vdots \\ \hat{p}_{i,L-1}(t) \\ \hat{p}_{i,L}(t) \end{pmatrix} = \kappa \begin{pmatrix} -1 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \end{pmatrix} \begin{pmatrix} \hat{p}_{i,1}(t) \\ \hat{p}_{i,2}(t) \\ \vdots \\ \hat{p}_{i,L-1}(t) \\ \hat{p}_{i,L}(t) \end{pmatrix} - \begin{pmatrix} b_1(t) \\ b_2(t) \\ \vdots \\ b_{L-1}(t) \\ b_L(t) \end{pmatrix} \tag{A.7.1}$$

with the same initial condition as in (A.5.2)

$$\hat{p}_{i,j}(0) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}.$$

Let $\Delta p_{i,j}(t) = p_{i,j}(t) - \hat{p}_{i,j}(t)$. From (A.5.1) and (A.7.1) we obtain

$$\frac{d}{dt}\begin{pmatrix} \Delta p_{i,1} \\ \Delta p_{i,2} \\ \vdots \\ \Delta p_{i,L-1} \\ \Delta p_{i,L} \end{pmatrix} = \kappa \begin{pmatrix} -1 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \end{pmatrix} \begin{pmatrix} \Delta p_{i,1} \\ \Delta p_{i,2} \\ \vdots \\ \Delta p_{i,L-1} \\ \Delta p_{i,L} \end{pmatrix} + \begin{pmatrix} b_1(t) \\ b_2(t) \\ \vdots \\ b_{L-1}(t) \\ b_L(t) \end{pmatrix}, \tag{A.7.2}$$

with initial condition $\Delta p_{i,j} = 0$, $j = 1, \ldots, L$.

The solution of (A.7.2) has the form

$$\begin{pmatrix} \Delta p_{i,1}(t) \\ \vdots \\ \Delta p_{i,L}(t) \end{pmatrix} = \int_0^t \mathbf{V} \begin{pmatrix} e^{\kappa\lambda_0(t-s)} & & \\ & \ddots & \\ & & e^{\kappa\lambda_{L-1}(t-s)} \end{pmatrix} \mathbf{V}^{-1} \begin{pmatrix} b_1(s) \\ \vdots \\ b_L(s) \end{pmatrix} ds, \tag{A.7.3}$$

where $\lambda_i$ is defined as (A.5.3) and $\mathbf{V}$ is defined as (A.5.6).

From (A.5.5) we can see that for any vector $\mathbf{p}(0) = (p_1(0), \ldots, p_L(0))$ whose elements are nonnegative, the following operation:

$$\mathbf{V} \begin{pmatrix} e^{\kappa\lambda_0 t} & & \\ & \ddots & \\ & & e^{\kappa\lambda_{L-1} t} \end{pmatrix} \mathbf{V}^{-1} \begin{pmatrix} p_1(0) \\ \vdots \\ p_L(0) \end{pmatrix}$$

returns a vector $(p_1(t), \ldots, p_L(t))^T$ which is also non-negative (every element in the vector is a probability value thus it should be non-negative). Now apply this observation to (A.7.3). Since $b_j(s) = a_j(s)\hat{p}_{i,j}(s) \geq 0$, it is evident that the overall expression in the integral in (A.7.3) is also nonnegative. Therefore the result $(\Delta p_{i,1}, \ldots, \Delta p_{i,L})^T$ is nonnegative as well, which implies $\Delta p_{i,j}(t) = p_{i,j}(t) - \hat{p}_{i,j}(t) \geq 0$.